

LinguaFrancaClocks

Frédéric Boulanger

April 2, 2020

Contents

1	Basic definitions	1
1.1	Periodic clocks	2
1.2	Sporadic clocks	2
2	Properties of clocks	3
3	Operations on clocks	4
4	Bounded clocks	9
4.1	Main theorem	13
5	Logical time	14
5.1	Chrono-periodic and chrono-sporadic clocks	15
6	Tests	18
theory	<i>LinguaFrancaClocks</i>	

imports *Main*

begin

1 Basic definitions

Instants are represented as the natural numbers. A clock represents an event that may occur or not at any instant. We model a clock as a function from `nat` to `bool`, which is `True` at every instant when the clock ticks (the event occurs).

type-synonym *clock* = $\langle nat \Rightarrow bool \rangle$

1.1 Periodic clocks

A clock is (k,p) -periodic if it ticks at instants separated by p instants, starting at instant k .

definition $kp\text{-periodic} :: \langle [nat, nat, clock] \Rightarrow bool \rangle$
where $\langle kp\text{-periodic } k \ p \ c \equiv$
 $(p > 0) \wedge (\forall n. c \ n = ((n \geq k) \wedge ((n - k) \bmod p = 0))) \rangle$

A 1-periodic clock always ticks starting at its offset

lemma $one\text{-periodic-ticks}$:
assumes $\langle kp\text{-periodic } k \ 1 \ c \rangle$
and $\langle n \geq k \rangle$
shows $\langle c \ n \rangle$
using $assms \ kp\text{-periodic-def} \text{ by } simp$

A p -periodic clock is a (k,p) -periodic clock starting from a given offset.

definition $\langle p\text{-periodic } p \ c \equiv (\exists k. kp\text{-periodic } k \ p \ c) \rangle$

lemma $p\text{-periodic-intro}[intro]$:
 $\langle kp\text{-periodic } k \ p \ c \implies p\text{-periodic } p \ c \rangle$
using $p\text{-periodic-def} \text{ by } blast$

No clock is 0-periodic.

lemma $no\text{-}0\text{-periodic}$:
 $\langle \neg p\text{-periodic } 0 \ c \rangle$
by $(simp \ add: \ kp\text{-periodic-def} \ p\text{-periodic-def})$

A periodic clock is a p -periodic clock for a given period.

definition $\langle periodic \ c \equiv (\exists p. p\text{-periodic } p \ c) \rangle$

lemma $periodic\text{-intro1}[intro]$:
 $\langle p\text{-periodic } p \ c \implies periodic \ c \rangle$
using $p\text{-periodic-def} \ periodic\text{-def} \text{ by } blast$

lemma $periodic\text{-intro2}[intro]$:
 $\langle kp\text{-periodic } k \ p \ c \implies periodic \ c \rangle$
using $p\text{-periodic-intro} \ periodic\text{-intro1} \text{ by } blast$

1.2 Sporadic clocks

A clock is p -sporadic if it ticks at instants separated at least by p instants.

definition $p\text{-sporadic} :: \langle [nat, clock] \Rightarrow bool \rangle$
where $\langle p\text{-sporadic } p \ c \equiv \forall t. c \ t \longrightarrow (\forall t'. (t' > t \wedge c \ t') \longrightarrow t' > t + p) \rangle$

Any clock is 0-sporadic

lemma $sporadic\text{-}0$: $\langle p\text{-sporadic } 0 \ c \rangle$
unfolding $p\text{-sporadic-def} \text{ by } auto$

We define sporadic clock as p-sporadic clocks for some non null interval p.

definition $\langle \text{sporadic } c \equiv (\exists p > 0. \text{ p-sporadic } p \ c) \rangle$

lemma *sporadic-intro*[intro]
 $\langle \llbracket \text{p-sporadic } p \ c; p > 0 \rrbracket \implies \text{sporadic } c \rangle$
using *sporadic-def* **by** *blast*

2 Properties of clocks

Some useful lemmas about modulo.

lemma *mod-sporadic*:
assumes $\langle (n::\text{nat}) \bmod p = 0 \rangle$
shows $\langle \forall n'. (n < n' \wedge n' < n+p) \longrightarrow \neg(n' \bmod p = 0) \rangle$
using *assms less-imp-add-positive* **by** *fastforce*

lemma *mod-sporadic'*:
assumes $\langle (n::\text{nat}) \bmod p = 0 \rangle$
shows $\langle \forall n'. (n < n' \wedge (n' \bmod p = 0)) \longrightarrow n' \geq n+p \rangle$
proof –
{ **fix** n' **assume** $h: \langle n < n' \wedge n' \bmod p = 0 \rangle$
hence $\langle n' \geq n+p \rangle$ **using** *mod-sporadic*[OF *assms*] **by** *auto*
} **thus** *?thesis* **by** *simp*
qed

lemma *mod-offset-sporadic*:
assumes $\langle (n::\text{nat}) \geq k \rangle$
and $\langle (n - k) \bmod p = 0 \rangle$
shows $\langle \forall n'. (n < n' \wedge n' < n+p) \longrightarrow \neg((n'-k) \bmod p = 0) \rangle$
proof –
from *assms* **have** $\langle \forall n'. n' > n \longrightarrow (n'-k) > (n-k) \rangle$ **by** (*simp add: diff-less-mono*)
with *mod-sporadic*[OF *assms*(2)] **show** *?thesis* **by** *auto*
qed

lemma *mod-offset-sporadic'*:
assumes $\langle (n::\text{nat}) \geq k \rangle$
and $\langle (n - k) \bmod p = 0 \rangle$
shows $\langle \forall n'. (n < n' \wedge ((n'-k) \bmod p = 0)) \longrightarrow n' \geq n+p \rangle$
proof –
from *assms* **have** $\langle \forall n'. n' > n \longrightarrow (n'-k) > (n-k) \rangle$ **by** (*simp add: diff-less-mono*)
with *mod-sporadic*[OF *assms*(2)] **show** *?thesis* **by** *auto*
qed

A (p+1)-periodic clock is p-sporadic.

lemma *periodic-suc-sporadic*:
assumes $\langle \text{p-periodic } (p+1) \ c \rangle$
shows $\langle \text{p-sporadic } p \ c \rangle$
proof –

from *assms* *p-periodic-def* **obtain** *k*
where $\langle kp\text{-periodic } k \text{ (Suc } p) \text{ } c \rangle$ **by** (*auto simp add: Suc-eq-plus1[symmetric]*)
hence $\langle \forall n. c \ n = ((n \geq k) \wedge ((n - k) \bmod (\text{Suc } p) = 0)) \rangle$ **unfolding**
kp-periodic-def **by** *simp*
thus *?thesis*
unfolding *p-sporadic-def*
using *mod-offset-sporadic'* [**where** *k=k* **and** *p=(Suc p)*]
by (*simp add: Suc-le-lessD*)
qed

3 Operations on clocks

The result of merging two clocks ticks whenever any of the two clocks ticks.

definition *merge* :: $\langle [clock, clock] \Rightarrow clock \rangle$ (**infix** $\langle \oplus \rangle$ 60)
where $\langle c1 \oplus c2 \equiv \lambda t. c1 \ t \vee c2 \ t \rangle$

lemma *merge-comm*: $\langle c \oplus c' = c' \oplus c \rangle$
by (*auto simp add: merge-def*)

Delaying a clock by one instant.

definition *delay* :: $\langle clock \Rightarrow clock \rangle$ (**infix** $\langle \$ \rangle$)
where $\langle \$c \ k = (\text{case } k \text{ of } 0 \Rightarrow \text{False} \mid \text{Suc } k' \Rightarrow c \ k') \rangle$

Sampling a clock with another clock.

definition *sampling* :: $\langle [clock, clock] \Rightarrow clock \rangle$ (**infix** $\langle \text{when} \rangle$ 70)
where $\langle c \ \text{when} \ c' \equiv \lambda k. c \ k \wedge c' \ k \rangle$

lemma *sampling-comm*: $\langle c \ \text{when} \ c' = c' \ \text{when} \ c \rangle$
by (*auto simp add: sampling-def*)

Merging two sporadic clocks does not necessary yields a sporadic clock.

lemma *merge-no-sporadic*:
 $\langle \exists c \ c'. \text{sporadic } c \wedge \text{sporadic } c' \wedge \neg \text{sporadic } (c \oplus c') \rangle$

proof –

define *c* :: *clock* **where** $\langle c = (\lambda t. t \bmod 2 = 0) \rangle$
define *c'* :: *clock* **where** $\langle c' = (\lambda t. t \geq 1 \wedge (t-1) \bmod 2 = 0) \rangle$

have $\langle p\text{-periodic } 2 \ c \rangle$ **unfolding** *p-periodic-def* *kp-periodic-def*
using *c-def* **by** *auto*

hence $1:\langle \text{sporadic } c \rangle$
using *periodic-suc-sporadic* *Suc-1[symmetric]* *sporadic-def* *zero-less-one*
by *auto*

have $\langle p\text{-periodic } 2 \ c' \rangle$ **unfolding** *p-periodic-def* *kp-periodic-def* **using** *c'-def*
by *auto*

hence $2:\langle \text{sporadic } c' \rangle$
using *periodic-suc-sporadic* *Suc-1[symmetric]* *sporadic-def* *zero-less-one*

```

    by auto

have  $\langle \neg \text{sporadic } (c \oplus c') \rangle$ 
proof -
  { assume  $\langle \text{sporadic } (c \oplus c') \rangle$ 
    from this obtain  $p$  where  $\ast : \langle p > 0 \rangle$  and  $\langle p\text{-sporadic } p \ (c \oplus c') \rangle$ 
    using sporadic-def by blast
    hence  $\langle \forall t. (c \oplus c') \ t \longrightarrow (\forall t'. (t < t' \wedge (c \oplus c') t') \longrightarrow t' > t + p) \rangle$ 
    by (simp add: p-sporadic-def)
    moreover have  $\langle (c \oplus c') \ 0 \rangle$  using c-def c'-def merge-def by simp
    moreover have  $\langle (c \oplus c') \ 1 \rangle$  using c-def c'-def merge-def by simp
    ultimately have False using  $\ast$  by blast
  } thus ?thesis ..
qed
with 1 and 2 show ?thesis by blast
qed

Delaying a periodic clock yields a shifted periodic clock.

lemma delay-shift-periodic:
  assumes  $\langle kp\text{-periodic } k \ p \ c \rangle$ 
  shows  $\langle kp\text{-periodic } (k+1) \ p \ (\$c) \rangle$ 
proof -
  from assms have 1:  $\langle p > 0 \rangle$  and 2:  $\langle (\forall n. c \ n = ((n \geq k) \wedge ((n - k) \bmod p = 0))) \rangle$ 
  unfolding kp-periodic-def by simp+
  have  $\langle \forall n. (\$c) \ n = (\text{case } n \text{ of } 0 \Rightarrow \text{False} \mid \text{Suc } n' \Rightarrow c \ n') \rangle$ 
  unfolding delay-def by simp
  with 2 have
    3:  $\langle \forall n. (\$c) \ n = (\text{case } n \text{ of } 0 \Rightarrow \text{False} \mid \text{Suc } n' \Rightarrow ((n' \geq k) \wedge ((n' - k) \bmod p = 0))) \rangle$ 
  by presburger
  have  $\langle \forall n. (\$c) \ n = ((n \geq k+1) \wedge ((n - (k+1)) \bmod p = 0)) \rangle$ 
  proof -
    { fix  $n$ 
      have  $\langle (\$c) \ n = ((n \geq k+1) \wedge ((n - (k+1)) \bmod p = 0)) \rangle$ 
      proof (cases  $n$ )
        case 0
        thus ?thesis by (simp add: 3)
      next
        case (Suc  $n'$ )
        with 3 have  $\langle (\$c) \ n = ((n' \geq k) \wedge ((n' - k) \bmod p = 0)) \rangle$  by simp
        also have  $\langle \dots = ((n-1 \geq k) \wedge ((n-1 - k) \bmod p = 0)) \rangle$  using Suc by
auto
        finally show ?thesis using Suc by fastforce
      }
    }
  qed
  } thus ?thesis ..
qed
thus ?thesis unfolding kp-periodic-def using 1 by simp
qed

```

Get the number of ticks on a clock from the beginning up to instant n .

definition $\text{ticks-up-to} :: \langle [\text{clock}, \text{nat}] \Rightarrow \text{nat} \rangle$
where $\langle \text{ticks-up-to } c \ n = \text{card } \{t. t \leq n \wedge c \ t\} \rangle$

There cannot be more than n event occurrences during n instants.

lemma $\langle \text{ticks-up-to } c \ n \leq \text{Suc } n \rangle$

proof –

have $\text{finite}: \langle \text{finite } \{t::\text{nat}. t \leq n\} \rangle$ **by** *simp*
have $\text{incl}: \langle \{t::\text{nat}. t \leq n \wedge c \ t\} \subseteq \{t::\text{nat}. t \leq n\} \rangle$ **by** *blast*
have $\langle \text{card } \{t::\text{nat}. t \leq n\} = \text{Suc } n \rangle$ **by** *simp*
with $\text{card-mono}[\text{OF } \text{finite } \text{incl}]$ **show** *?thesis* **unfolding** ticks-up-to-def **by** *simp*
qed

Counting event occurrences.

definition $\langle \text{count } b \ n \equiv \text{if } b \text{ then } \text{Suc } n \text{ else } n \rangle$

The count of event occurrences cannot grow by more than one at each instant.

lemma $\text{count-inc}: \langle \text{count } b \ n \leq \text{Suc } n \rangle$
using count-def **by** *simp*

Alternative definition of the number of event occurrences using fold.

definition $\text{ticks-up-to-fold} :: \langle [\text{clock}, \text{nat}] \Rightarrow \text{nat} \rangle$
where $\langle \text{ticks-up-to-fold } c \ n = \text{fold count } (\text{map } c \ [0..<\text{Suc } n]) \ 0 \rangle$

Alternative definition of the number of event occurrences as a function.

fun $\text{ticks-up-to-fun} :: \langle [\text{clock}, \text{nat}] \Rightarrow \text{nat} \rangle$
where
 $\langle \text{ticks-up-to-fun } c \ 0 = \text{count } (c \ 0) \ 0 \rangle$
 $\mid \langle \text{ticks-up-to-fun } c \ (\text{Suc } n) = \text{count } (c \ (\text{Suc } n)) \ (\text{ticks-up-to-fun } c \ n) \rangle$

Proof that the original definition and the function definition are equivalent.
Use this to generate code.

lemma $\text{ticks-up-to-is-fun}[\text{code}]: \langle \text{ticks-up-to } c \ n = \text{ticks-up-to-fun } c \ n \rangle$

proof (*induction n*)

case 0

have $\langle \text{ticks-up-to } c \ 0 = \text{card } \{t. t \leq 0 \wedge c \ t\} \rangle$
by (*simp add: ticks-up-to-def*)
also have $\langle \dots = \text{card } \{t. t=0 \wedge c \ t\} \rangle$ **by** *simp*
also have $\langle \dots = (\text{if } c \ 0 \text{ then } 1 \text{ else } 0) \rangle$
by (*simp add: Collect-conv-if*)
also have $\langle \dots = \text{ticks-up-to-fun } c \ 0 \rangle$
using $\text{ticks-up-to-fun.simps}(1)$ count-def **by** *simp*
finally show *?case* .

next

case $(\text{Suc } n)$
show *?case*

```

proof (cases ⟨c (Suc n)⟩)
  case True
    hence ⟨{t. t ≤ Suc n ∧ c t} = insert (Suc n) {t. t ≤ n ∧ c t}⟩ by auto
    hence ⟨ticks-up-to c (Suc n) = Suc (ticks-up-to c n)⟩
      by (simp add: ticks-up-to-def)
    also have ⟨... = Suc (ticks-up-to-fun c n)⟩ using Suc.IH by simp
    finally show ?thesis by (simp add: count-def ⟨c (Suc n)⟩)
  next
    case False
      hence ⟨{t. t ≤ Suc n ∧ c t} = {t. t ≤ n ∧ c t}⟩ using le-Suc-eq by blast
      hence ⟨ticks-up-to c (Suc n) = ticks-up-to c n⟩
        by (simp add: ticks-up-to-def)
      also have ⟨... = ticks-up-to-fun c n⟩ using Suc.IH by simp
      finally show ?thesis by (simp add: count-def ⟨¬c (Suc n)⟩)
    qed
  qed

Proof that the original definition and the definition using fold are equivalent.

lemma ticks-up-to-is-fold: ⟨ticks-up-to c n = ticks-up-to-fold c n⟩
proof (induction n)
  case 0
    have ⟨ticks-up-to c 0 = card {t. t ≤ 0 ∧ c t}⟩
      by (simp add: ticks-up-to-def)
    also have ⟨... = card {t. t=0 ∧ c t}⟩ by simp
    also have ⟨... = (if c 0 then 1 else 0)⟩
      by (simp add: Collect-conv-if)
    also have ⟨... = ticks-up-to-fold c 0⟩
      using ticks-up-to-fold-def count-def by simp
    finally show ?case .
  next
    case (Suc n)
      show ?case
      proof (cases ⟨c (Suc n)⟩)
        case True
          hence ⟨{t. t ≤ Suc n ∧ c t} = insert (Suc n) {t. t ≤ n ∧ c t}⟩ by auto
          hence ⟨ticks-up-to c (Suc n) = Suc (ticks-up-to c n)⟩
            by (simp add: ticks-up-to-def)
          also have ⟨... = Suc (ticks-up-to-fold c n)⟩ using Suc.IH by simp
          finally show ?thesis by (simp add: ticks-up-to-fold-def count-def ⟨c (Suc
n)⟩)
        next
          case False
            hence ⟨{t. t ≤ Suc n ∧ c t} = {t. t ≤ n ∧ c t}⟩ using le-Suc-eq by blast
            hence ⟨ticks-up-to c (Suc n) = ticks-up-to c n⟩
              by (simp add: ticks-up-to-def)
            also have ⟨... = ticks-up-to-fold c n⟩ using Suc.IH by simp
            finally show ?thesis by (simp add: ticks-up-to-fold-def count-def ⟨¬c (Suc
n)⟩)
          qed
        qed
      
```

qed

Number of ticks during an n instant window starting at k_0 .

definition *tick-count* :: $\langle [clock, nat, nat] \Rightarrow nat \rangle$
where $\langle tick_count\ c\ k_0\ n \equiv card\ \{k. k_0 \leq k \wedge k < k_0 + n \wedge c\ k\} \rangle$

The number of ticks is monotonous with regard to the window width.

lemma *tick-count-mono*:

assumes $\langle n' \geq n \rangle$

shows $\langle tick_count\ c\ t_0\ n' \geq tick_count\ c\ t_0\ n \rangle$

proof –

have *finite*: $\langle finite\ \{t::nat. t_0 \leq t \wedge t < t_0 + n' \wedge c\ t\} \rangle$ **by** *simp*

from *assms* **have** *incl*:

$\langle \{t::nat. t_0 \leq t \wedge t < t_0 + n \wedge c\ t\} \subseteq \{t::nat. t_0 \leq t \wedge t < t_0 + n' \wedge c\ t\} \rangle$ **by** *auto*

have $\langle card\ \{t::nat. t_0 \leq t \wedge t < t_0 + n \wedge c\ t\} \leq card\ \{t::nat. t_0 \leq t \wedge t < t_0 + n' \wedge c\ t\} \rangle$

using *card-mono[OF finite incl]* .

thus *?thesis* **using** *tick-count-def* **by** *simp*

qed

The interval $[t, t+n[$ contains n instants.

lemma *card-interval*: $\langle card\ \{t. t_0 \leq t \wedge t < t_0 + n\} = n \rangle$

proof (*induction n*)

case 0

then show *?case* **by** *simp*

next

case (*Suc n*)

have $\langle \{t. t_0 \leq t \wedge t < t_0 + (Suc\ n)\} = insert\ (t_0 + n)\ \{t. t_0 \leq t \wedge t < t_0 + n\} \rangle$

by *auto*

hence $\langle card\ \{t. t_0 \leq t \wedge t < t_0 + (Suc\ n)\} = Suc\ (card\ \{t. t_0 \leq t \wedge t < t_0 + n\}) \rangle$

by *simp*

with *Suc.IH* **show** *?case* **by** *simp*

qed

There cannot be more than n occurrences of an event in an interval of n instants.

lemma *tick-count-bound*: $\langle tick_count\ c\ t_0\ n \leq n \rangle$

proof –

have *finite*: $\langle finite\ \{t. t_0 \leq t \wedge t < t_0 + n\} \rangle$ **by** *simp*

have *incl*: $\langle \{t. t_0 \leq t \wedge t < t_0 + n \wedge c\ t\} \subseteq \{t. t_0 \leq t \wedge t < t_0 + n\} \rangle$ **by** *blast*

show *?thesis* **using** *tick-count-def card-interval card-mono[OF finite incl]* **by** *simp*

qed

No event occurrence occur in 0 instant.

lemma *tick-count-0[code]*: $\langle tick_count\ c\ t_0\ 0 = 0 \rangle$

unfolding *tick-count-def* **by** *simp*

Event occurrences starting from instant 0 are event occurrences from the beginning.

lemma *tick-count-orig*[code]:
 $\langle \text{tick-count } c \ 0 \ (\text{Suc } n) = \text{ticks-up-to } c \ n \rangle$
unfolding *tick-count-def ticks-up-to-def*
using *less-Suc-eq-le* **by** *simp*

Counting event occurrences between two instants is simply subtracting occurrence counts from the beginning.

lemma *tick-count-diff*[code]:
 $\langle \text{tick-count } c \ (\text{Suc } t_0) \ n = (\text{ticks-up-to } c \ (t_0+n)) - (\text{ticks-up-to } c \ t_0) \rangle$
proof –
have *incl*: $\langle \{t. t \leq t_0 \wedge c \ t\} \subseteq \{t. t \leq t_0+n \wedge c \ t\} \rangle$ **by** *auto*
have $\langle \{t. (\text{Suc } t_0) \leq t \wedge t < (\text{Suc } t_0)+n \wedge c \ t\}$
 $= \{t. t \leq t_0+n \wedge c \ t\} - \{t. t \leq t_0 \wedge c \ t\} \rangle$ **by** *auto*
hence $\langle \text{card } \{t. (\text{Suc } t_0) \leq t \wedge t < (\text{Suc } t_0)+n \wedge c \ t\}$
 $= \text{card } \{t. t \leq t_0+n \wedge c \ t\} - \text{card } \{t. t \leq t_0 \wedge c \ t\} \rangle$
by (*simp add: card-Diff-subset incl*)
thus *?thesis* **unfolding** *tick-count-def ticks-up-to-def* .
qed

The merge of two clocks has less ticks than the union of the ticks of the two clocks.

lemma *tick-count-merge*:
 $\langle \text{tick-count } (c \oplus c') \ t_0 \ n \leq \text{tick-count } c \ t_0 \ n + \text{tick-count } c' \ t_0 \ n \rangle$
proof –
have $\langle \{t::\text{nat}. t_0 \leq t \wedge t < t_0+n \wedge ((c \oplus c') \ t)\}$
 $= \{t::\text{nat}. t_0 \leq t \wedge t < t_0+n \wedge c \ t\} \cup \{t::\text{nat}. t_0 \leq t \wedge t < t_0+n \wedge c' \ t\} \rangle$
using *merge-def* **by** *auto*
hence $\langle \text{card } \{t::\text{nat}. t_0 \leq t \wedge t < t_0+n \wedge ((c \oplus c') \ t)\}$
 $\leq \text{card } \{t::\text{nat}. t_0 \leq t \wedge t < t_0+n \wedge c \ t\}$
 $+ \text{card } \{t::\text{nat}. t_0 \leq t \wedge t < t_0+n \wedge c' \ t\} \rangle$ **by** (*simp add: card-Un-le*)
thus *?thesis* **unfolding** *tick-count-def* .
qed

4 Bounded clocks

An (n,m)-bounded clock does not tick more than m times in a n interval of width n.

definition *bounded* :: $\langle [\text{nat}, \text{nat}, \text{clock}] \Rightarrow \text{bool} \rangle$
where $\langle \text{bounded } n \ m \ c \equiv \forall t. \text{tick-count } c \ t \ n \leq m \rangle$

All clocks are (n,n)-bounded.

lemma *bounded-n*: $\langle \text{bounded } n \ n \ c \rangle$
unfolding *bounded-def* **using** *tick-count-bound* **by** (*simp add: le-imp-less-Suc*)

A sporadic clock is bounded.

lemma *spor-bound*:

assumes $\langle \forall t::nat. c\ t \longrightarrow (\forall t'. (t < t' \wedge t' \leq t+n) \longrightarrow \neg(c\ t')) \rangle$

shows $\langle \forall t::nat. \text{card } \{t'. t \leq t' \wedge t' \leq t+n \wedge c\ t'\} \leq 1 \rangle$

proof –

{ **fix** $t::nat$

have $\langle \text{card } \{t'. t \leq t' \wedge t' \leq t+n \wedge c\ t'\} \leq 1 \rangle$

proof (*cases* $\langle c\ t \rangle$)

case *True*

with *assms* **have** $\langle \forall t'. (t < t' \wedge t' \leq t+n) \longrightarrow \neg(c\ t') \rangle$ **by** *simp*

hence *empty*: $\langle \text{card } \{t'. t < t' \wedge t' \leq t+n \wedge c\ t'\} = 0 \rangle$ **by** *simp*

have *finite*: $\langle \text{finite } \{t'. t < t' \wedge t' \leq t+n \wedge c\ t'\} \rangle$ **by** *simp*

have *notin*: $\langle t \notin \{t'. t < t' \wedge t' \leq t+n \wedge c\ t'\} \rangle$ **by** *simp*

have $\langle \{t'. t \leq t' \wedge t' \leq t+n \wedge c\ t'\} \rangle$

$= \text{insert } t\ \{t'. t < t' \wedge t' \leq t+n \wedge c\ t'\}$ **using** $\langle c\ t \rangle$ **by** *auto*

hence $\langle \text{card } \{t'. t \leq t' \wedge t' \leq t+n \wedge c\ t'\} = 1 \rangle$

using *empty card-insert-disjoint[OF finite notin]* **by** *simp*

then show *?thesis* **by** *simp*

next

case *False*

then show *?thesis*

proof(*cases* $\langle \exists tt. t < tt \wedge tt \leq t+n \wedge c\ tt \rangle$)

case *True*

hence $\langle \exists ttmin. t < ttmin \wedge ttmin \leq t+n \wedge c\ ttmin \wedge (\forall tt'. (t < tt' \wedge tt' \leq t+n \wedge c\ tt') \longrightarrow ttmin \leq tt') \rangle$

by (*metis add-lessD1 add-less-mono1 assms le-eq-less-or-eq le-refl less-imp-le-nat nat-le-iff-add nat-le-linear*)

from this obtain *ttmin* **where**

tmin: $\langle t < ttmin \wedge ttmin \leq t+n \wedge c\ ttmin \wedge (\forall tt'. (t < tt' \wedge tt' \leq t+n \wedge c\ tt') \longrightarrow ttmin \leq tt') \rangle$ **by** *blast*

hence *tick*: $\langle c\ ttmin \rangle$ **by** *simp*

with *assms* **have** *notick*: $\langle (\forall t'. ttmin < t' \wedge t' \leq ttmin + n \longrightarrow \neg c\ t') \rangle$

by *simp*

have $\langle \forall t'. (t < t' \wedge t' < ttmin) \longrightarrow \neg c\ t' \rangle$ **using** *tmin* $\langle \neg c\ t \rangle$ **by** *auto*

moreover from *notick tmin* **have**

$\langle \forall t'. (ttmin < t' \wedge t' \leq t+n) \longrightarrow \neg c\ t' \rangle$ **by** *auto*

ultimately have $\langle \forall t'::nat. (t \leq t' \wedge t' \leq t+n \wedge c\ t') \longrightarrow t' = ttmin \rangle$

using *tick tmin* $\langle \neg c\ t \rangle$ *le-eq-less-or-eq* **by** *auto*

hence $\langle \{t'. t \leq t' \wedge t' \leq t+n \wedge c\ t'\} = \{ttmin\} \rangle$ **using** *tmin* **by** *fastforce*

hence $\langle \text{card } \{t'. t \leq t' \wedge t' \leq t+n \wedge c\ t'\} = 1 \rangle$ **by** *simp*

thus *?thesis* **by** *simp*

next

case *False*

with $\langle \neg c\ t \rangle$ **have** $\langle \forall t'. t \leq t' \wedge t' \leq t+n \longrightarrow \neg c\ t' \rangle$

using *nat-less-le* **by** *blast*

hence $\langle \text{card } \{t'. t \leq t' \wedge t' \leq t+n \wedge c\ t'\} = 0 \rangle$ **by** *simp*

thus *?thesis* **by** *linarith*

qed

qed

} thus ?thesis ..
qed

A sporadic clock is bounded.

lemma *spor-bound'*:

assumes $\langle \forall t::nat. c\ t \longrightarrow (\forall t'. (t < t' \wedge c\ t') \longrightarrow t' > t+n) \rangle$

shows $\langle \forall t::nat. card\ \{t'. t \leq t' \wedge t' \leq t+n \wedge c\ t'\} \leq 1 \rangle$

proof –

{ **fix** $t::nat$

have $\langle card\ \{t'. t \leq t' \wedge t' \leq t+n \wedge c\ t'\} \leq 1 \rangle$

proof (*cases* $\langle c\ t \rangle$)

case *True*

with *assms* **have** $\langle \forall t'. (t < t' \wedge c\ t') \longrightarrow t' > t+n \rangle$ **by** *simp*

hence *empty*: $\langle card\ \{t'. t < t' \wedge t' \leq t+n \wedge c\ t'\} = 0 \rangle$ **by** *auto*

have *finite*: $\langle finite\ \{t'. t < t' \wedge t' \leq t+n \wedge c\ t'\} \rangle$ **by** *simp*

have *notin*: $\langle t \notin \{t'. t < t' \wedge t' \leq t+n \wedge c\ t'\} \rangle$ **by** *simp*

have $\langle \{t'. t \leq t' \wedge t' \leq t+n \wedge c\ t'\} =$

$\text{insert } t\ \{t'. t < t' \wedge t' \leq t+n \wedge c\ t'\} \rangle$ **using** $\langle c\ t \rangle$ **by** *auto*

hence $\langle card\ \{t'. t \leq t' \wedge t' \leq t+n \wedge c\ t'\} = 1 \rangle$

using *empty card-insert-disjoint[OF finite notin]* **by** *simp*

then show ?thesis **by** *simp*

next

case *False*

then show ?thesis

proof(*cases* $\langle \exists tt. t < tt \wedge tt \leq t+n \wedge c\ tt \rangle$)

case *True*

hence $\langle \exists ttmin. t < ttmin \wedge ttmin \leq t+n \wedge c\ ttmin$

$\wedge (\forall tt'. (t < tt' \wedge tt' \leq t+n \wedge c\ tt') \longrightarrow ttmin \leq tt') \rangle$

by (*metis add-lessD1 add-less-mono1 assms le-Suc-ex le-eq-less-or-eq le-refl*

less-imp-le-nat nat-le-linear nat-neq-iff)

from this obtain *ttmin* **where**

tmin: $\langle t < ttmin \wedge ttmin \leq t+n \wedge c\ ttmin$

$\wedge (\forall tt'. (t < tt' \wedge tt' \leq t+n \wedge c\ tt') \longrightarrow ttmin \leq tt') \rangle$ **by** *blast*

hence *tick*: $\langle c\ ttmin \rangle$ **by** *simp*

with *assms* **have** *notick*: $\langle (\forall t'. ttmin < t' \wedge c\ t' \longrightarrow t' > ttmin + n) \rangle$ **by**

simp

have $\langle \forall t'. (t < t' \wedge t' < ttmin) \longrightarrow \neg c\ t' \rangle$ **using** *tmin* $\langle \neg c\ t \rangle$ **by** *auto*

moreover from *notick* *tmin* **have**

$\langle \forall t'. (ttmin < t' \wedge t' \leq t+n) \longrightarrow \neg c\ t' \rangle$ **by** *auto*

ultimately have $\langle \forall t'::nat. (t \leq t' \wedge t' \leq t+n \wedge c\ t') \longrightarrow t' = ttmin \rangle$

using *tick* *tmin* $\langle \neg c\ t \rangle$ *le-eq-less-or-eq* **by** *auto*

hence $\langle \{t'. t \leq t' \wedge t' \leq t+n \wedge c\ t'\} = \{ttmin\} \rangle$ **using** *tmin* **by** *fastforce*

hence $\langle card\ \{t'. t \leq t' \wedge t' \leq t+n \wedge c\ t'\} = 1 \rangle$ **by** *simp*

thus ?thesis **by** *simp*

next

case *False*

with $\langle \neg c\ t \rangle$ **have** $\langle \forall t'. t \leq t' \wedge t' \leq t+n \longrightarrow \neg c\ t' \rangle$

using *nat-less-le* **by** *blast*

hence $\langle \text{card } \{t'. t \leq t' \wedge t' \leq t+n \wedge c\ t'\} = 0 \rangle$ by *simp*
 thus ?thesis by *linarith*
 qed
 qed
 } thus ?thesis ..
 qed

An n -sporadic clock is $(n+1, 1)$ -bounded.

lemma *spor-bounded*:

assumes $\langle p\text{-sporadic } n\ c \rangle$
 shows $\langle \text{bounded } (n+1)\ 1\ c \rangle$
proof –
 from *assms* have $\langle \forall t. c\ t \longrightarrow (\forall t'. (t < t' \wedge c\ t') \longrightarrow t' > t+n) \rangle$
 using *p-sporadic-def* by *simp*
 from *spor-bound* [OF *this*] have $\langle \forall t. \text{card } \{t'. t \leq t' \wedge t' \leq t+n \wedge c\ t'\} \leq 1 \rangle$.
 hence $\langle \forall t. \text{card } \{t'. t \leq t' \wedge t' < \text{Suc } (t+n) \wedge c\ t'\} \leq 1 \rangle$
 using *less-Suc-eq-le* by *auto*
 hence $\langle \forall t. \text{card } \{t'. t \leq t' \wedge t' < t + \text{Suc } n \wedge c\ t'\} \leq 1 \rangle$ by *auto*
 thus ?thesis unfolding *bounded-def tick-count-def Suc-eq-plus1* .
 qed

An n -sporadic clock is $(n+2, 2)$ -bounded.

lemma *spor-bounded2*:

assumes $\langle p\text{-sporadic } n\ c \rangle$
 shows $\langle \text{bounded } (n+2)\ 2\ c \rangle$
proof –
 from *spor-bounded* [OF *assms*] have
 *: $\langle \forall t. \text{card } \{t'. t \leq t' \wedge t' < t + \text{Suc } n \wedge c\ t'\} \leq 1 \rangle$
 unfolding *bounded-def tick-count-def* by *simp*
 hence $\langle \forall t. \text{card } \{t'. t \leq t' \wedge t' < \text{Suc } (t + \text{Suc } n) \wedge c\ t'\} \leq \text{Suc } 1 \rangle$
proof –
 { fix $t::\text{nat}$
 from * have **: $\langle \text{card } \{t'. t \leq t' \wedge t' < t + \text{Suc } n \wedge c\ t'\} \leq 1 \rangle$ by *simp*
 have $\langle \text{card } \{t'. t \leq t' \wedge t' < \text{Suc } (t + \text{Suc } n) \wedge c\ t'\} \leq \text{Suc } 1 \rangle$
proof (cases $\langle c\ (t + \text{Suc } n) \rangle$)
 case *True*
 hence $\langle \{t'. t \leq t' \wedge t' < \text{Suc } (t + \text{Suc } n) \wedge c\ t'\} \rangle$
 = $\text{insert } (t+\text{Suc } n)\ \{t'. t \leq t' \wedge t' < t + \text{Suc } n \wedge c\ t'\}$ by *auto*
 hence $\langle \text{card } \{t'. t \leq t' \wedge t' < \text{Suc } (t + \text{Suc } n) \wedge c\ t'\} \rangle$
 = $\text{Suc } (\text{card } \{t'. t \leq t' \wedge t' < t + \text{Suc } n \wedge c\ t'\})$ by *simp*
 thus ?thesis using ** by *simp*
 next
 case *False*
 hence $\langle \{t'. t \leq t' \wedge t' < \text{Suc } (t + \text{Suc } n) \wedge c\ t'\} \rangle$
 = $\{t'. t \leq t' \wedge t' < t + \text{Suc } n \wedge c\ t'\}$ using *less-Suc-eq* by *blast*
 hence $\langle \text{card } \{t'. t \leq t' \wedge t' < \text{Suc } (t + \text{Suc } n) \wedge c\ t'\} \rangle$
 = $(\text{card } \{t'. t \leq t' \wedge t' < t + \text{Suc } n \wedge c\ t'\})$ by *simp*
 thus ?thesis using ** by *simp*
 qed
 }
 qed

```

    } thus ?thesis ..
qed
thus ?thesis unfolding bounded-def tick-count-def
  by (metis Suc-1 add-Suc-right Suc-eq-plus1)
qed

```

A bounded clock on an interval is also bounded on a narrower interval.

```

lemma bounded-less:
  assumes ⟨bounded n' m c⟩
    and ⟨n' ≥ n⟩
  shows ⟨bounded n m c⟩
  using assms(1) unfolding bounded-def
  using tick-count-mono[OF assms(2)] order-trans by blast

```

The merge of two bounded clocks is bounded.

```

lemma bounded-merge:
  assumes ⟨bounded n m c⟩
    and ⟨bounded n' m' c'⟩
    and ⟨n' ≥ n⟩
  shows ⟨bounded n (m+m') (c⊕c')⟩
  using tick-count-merge bounded-less[OF assms(2,3)] assms(1,2) add-mono order-trans
  unfolding bounded-def by blast

```

The merge of two sporadic clocks is bounded.

```

lemma sporadic-bounded1:
  assumes ⟨p-sporadic n c⟩
    and ⟨p-sporadic n' c'⟩
    and ⟨n' ≥ n⟩
  shows ⟨bounded (n+1) 2 (c⊕c')⟩
proof -
  have 1:⟨bounded (n+1) 1 c⟩ using spor-bounded[OF assms(1)] .
  have 2:⟨bounded (n'+1) 1 c'⟩ using spor-bounded[OF assms(2)] .
  from assms(3) have 3:⟨n'+1 ≥ n+1⟩ by simp
  have ⟨1+1 = (2::nat)⟩ by simp
  with bounded-merge[OF 1 2 3] show ?thesis by metis
qed

```

4.1 Main theorem

The merge of two sporadic clocks is bounded on the min of the bounding intervals.

```

theorem sporadic-bounded-min:
  assumes ⟨p-sporadic n c⟩
    and ⟨p-sporadic n' c'⟩
  shows ⟨bounded ((min n n')+1) 2 (c⊕c')⟩
proof (cases ⟨n ≤ n'⟩)
  case True
  hence ⟨min n n' = n⟩ by simp

```

```

    thus ?thesis using sporadic-bounded1[OF assms True] by simp
next
  case False
    hence 1:( $n' = \min n \ n'$ ) and 2:( $n' \leq n$ ) by simp+
    from sporadic-bounded1[OF assms(2) assms(1) 2] 1 show ?thesis using
merge-comm by simp
qed

```

```

end
theory LinguaFrancaLogicalTime

```

```

imports LinguaFrancaClocks

```

```

begin

```

5 Logical time

Logical time is a natural number that is attached to instants. Logical time can stay constant for an arbitrary number of instants, but it cannot decrease. When logical time stays constant for an infinite number of instants, we have a Zeno condition.

```

typedef time = ⟨ $t::nat \Rightarrow nat$ . mono  $t$ ⟩
  using mono-Suc by blast

```

```

setup-lifting type-definition-time

```

A chronometric clock is a clock associated with a time line.

```

type-synonym chronoclock = ⟨clock × time⟩

```

@term $c \nabla t$ tells whether chronometric clock c ticks at instant t .

```

definition ticks :: ⟨chronoclock, nat⟩  $\Rightarrow$  bool (infix ⟨ $\nabla$ ⟩ 60)
  where ⟨ $c \nabla t \equiv (fst\ c)\ t$ ⟩

```

@term c_t is the logical time on clock c at instant t .

```

lift-definition time-at :: ⟨chronoclock, nat⟩  $\Rightarrow$  nat (⟨ $\cdot$ ⟩ [60, 60])
  is ⟨ $\lambda c\ t. (snd\ c)\ t$ ⟩ .

```

```

lemmas chronoclocks-simp[simp] = ticks-def time-at-def

```

As consequence of the definition of the *time* type, (∇) is monotonous for any clock.

```

lemma mono-chronotime:
  ⟨mono (time-at  $c$ )⟩ using Rep-time by auto

```

An event occurs at a given time if the clock ticks at some instant at that time.

definition $\text{occurs} :: \langle [\text{nat}, \text{chronoclock}] \Rightarrow \text{bool} \rangle$
where $\langle \text{occurs } n \ c \equiv \exists k. (c \ \nabla \ k \wedge c_k = n) \rangle$

An event occurs once at a given time if the clock ticks at exactly one instant at that time.

definition $\text{occurs-once} :: \langle [\text{nat}, \text{chronoclock}] \Rightarrow \text{bool} \rangle$
where $\langle \text{occurs-once } n \ c \equiv \exists !k. (c \ \nabla \ k \wedge c_k = n) \rangle$

lemma $\text{occurs-once-occurs}:$
 $\langle \text{occurs-once } n \ c \implies \text{occurs } n \ c \rangle$

unfolding $\text{occurs-once-def occurs-def by blast}$

A clock is strict at a given time if it ticks at most once at that time.

definition $\text{strict-at} :: \langle [\text{nat}, \text{chronoclock}] \Rightarrow \text{bool} \rangle$
where $\langle \text{strict-at } n \ c \equiv (\text{occurs } n \ c \longrightarrow \text{occurs-once } n \ c) \rangle$

definition $\text{strict-clock} :: \langle \text{chronoclock} \Rightarrow \text{bool} \rangle$
where $\langle \text{strict-clock } c \equiv (\forall n. \text{strict-at } n \ c) \rangle$

5.1 Chrono-periodic and chrono-sporadic clocks

The introduction of logical time allows us to define periodicity and sporadicity on logical time instead of instant index.

definition $\text{kp-chronoperiodic} :: \langle [\text{nat}, \text{nat}, \text{chronoclock}] \Rightarrow \text{bool} \rangle$
where $\langle \text{kp-chronoperiodic } k \ p \ c \equiv (p > 0) \wedge (\forall n. \text{occurs } n \ c = ((n \geq k) \wedge ((n - k) \bmod p = 0))) \rangle$

definition $\text{p-chronoperiodic} :: \langle [\text{nat}, \text{chronoclock}] \Rightarrow \text{bool} \rangle$
where $\langle \text{p-chronoperiodic } p \ c \equiv \exists k. \text{kp-chronoperiodic } k \ p \ c \rangle$

definition $\text{chronoperiodic} :: \langle [\text{chronoclock}] \Rightarrow \text{bool} \rangle$
where $\langle \text{chronoperiodic } c \equiv \exists p. \text{p-chronoperiodic } p \ c \rangle$

A clock is strictly chronoperiodic if it ticks only once at the logical times when it ticks.

definition $\text{chronoperiodic-strict} :: \langle [\text{chronoclock}] \Rightarrow \text{bool} \rangle$
where $\langle \text{chronoperiodic-strict } c \equiv \text{chronoperiodic } c \wedge \text{strict-clock } c \rangle$

definition $\text{p-chronoperiodic-strict} :: \langle [\text{nat}, \text{chronoclock}] \Rightarrow \text{bool} \rangle$
where $\langle \text{p-chronoperiodic-strict } p \ c \equiv \text{p-chronoperiodic } p \ c \wedge \text{strict-clock } c \rangle$

lemma $\langle \text{chronoperiodic-strict } c \implies \text{chronoperiodic } c \rangle$
unfolding $\text{chronoperiodic-strict-def by simp}$

definition $\text{p-chronosporadic} :: \langle [\text{nat}, \text{chronoclock}] \Rightarrow \text{bool} \rangle$
where $\langle \text{p-chronosporadic } p \ c \equiv$
 $\forall t. \text{occurs } t \ c \longrightarrow (\forall t'. (t' > t \wedge \text{occurs } t' \ c) \longrightarrow t' > t + p) \rangle$

definition $\langle p\text{-chronosporadic-strict } p \ c \equiv p\text{-chronosporadic } p \ c \wedge \text{strict-clock } c \rangle$

definition $\langle \text{chronosporadic } c \equiv (\exists p > 0. \ p\text{-chronosporadic } p \ c) \rangle$

definition $\langle \text{chronosporadic-strict } c \equiv \text{chronosporadic } c \wedge \text{strict-clock } c \rangle$

lemma *chrono-periodic-suc-sporadic*:

assumes $\langle p\text{-chronoperiodic } (p+1) \ c \rangle$

shows $\langle p\text{-chronosporadic } p \ c \rangle$

proof –

from *assms* $p\text{-chronoperiodic-def}$ **obtain** k

where $\langle kp\text{-chronoperiodic } k \ (p+1) \ c \rangle$ **by** *blast*

hence $\ast: \langle \forall n. \text{occurs } n \ c = ((n \geq k) \wedge ((n - k) \bmod (p+1) = 0)) \rangle$

unfolding $kp\text{-chronoperiodic-def}$ **by** *simp*

with *mod-offset-sporadic* $[\text{of } k - (p+1)]$ **have**

$\langle \forall n. \text{occurs } n \ c \longrightarrow (\forall n'. (n < n' \wedge ((n' - k) \bmod (p+1) = 0)) \longrightarrow n' \geq n + p + 1) \rangle$

by *simp*

thus *?thesis* **unfolding** $p\text{-chronosporadic-def}$ **by** (*simp add: Suc-le-lessD*)

qed

lemma *chrono-periodic-suc-sporadic-strict*:

assumes $\langle p\text{-chronoperiodic-strict } (p+1) \ c \rangle$

shows $\langle p\text{-chronosporadic-strict } p \ c \rangle$

using *assms chrono-periodic-suc-sporadic*

$p\text{-chronoperiodic-strict-def } p\text{-chronosporadic-strict-def}$

by *simp*

Number of ticks up to a given logical time. This counts distinct ticks that happen at the same logical time.

definition *chrono-dense-up-to* $:: \langle [\text{chronoclock}, \text{nat}] \Rightarrow \text{nat} \rangle$

where $\langle \text{chrono-dense-up-to } c \ n = \text{card } \{t. c_t \leq n \wedge c \nabla t\} \rangle$

A clock is Zeno if it ticks an infinite number of times in a finite amount of time.

definition *zeno-clock* $:: \langle \text{chronoclock} \Rightarrow \text{bool} \rangle$

where $\langle \text{zeno-clock } c \equiv (\exists \omega. \text{infinite } \{t. c_t \leq \omega \wedge c \nabla t\}) \rangle$

Number of occurrences of an event up to a given logical time. This does not count separately ticks that occur at the same logical time.

definition *chrono-up-to* $:: \langle [\text{chronoclock}, \text{nat}] \Rightarrow \text{nat} \rangle$

where $\langle \text{chrono-up-to } c \ n = \text{card } \{t. t \leq n \wedge \text{occurs } t \ c\} \rangle$

lemma *chrono-up-to-bounded*:

$\langle \text{chrono-up-to } c \ n \leq n + 1 \rangle$

proof –

have $\text{bound}: \langle \text{card } \{t. t \leq n\} = n + 1 \rangle$ **using** *Suc-eq-plus1 card-Collect-le-nat* **by** *simp*

hence $\text{finite}:\langle \text{finite } \{t. t \leq n\} \rangle$ **by** *simp*
have $\text{included}:\langle \{t. t \leq n \wedge \text{occurs } t \ c\} \subseteq \{t. t \leq n\} \rangle$ **by** *blast*
from $\text{card-mono}[OF \ \text{finite included}]$ *bound*
show *?thesis* **unfolding** *chrono-up-to-def* **by** *simp*
qed

For any time n , a non Zeno clock has less occurrences than ticks up to n . This is also true for Zeno clock, but we count ticks and occurrences using *card*, and in Isabelle/HOL, the cardinal of an infinite set is 0, so the inequality breaks when there are infinitely many ticks before a given time.

lemma *not-zeno-sparse*:

assumes $\langle \neg \text{zeno-clock } c \rangle$
shows $\langle \text{chrono-up-to } c \ n \leq \text{chrono-dense-up-to } c \ n \rangle$
proof –
from *assms* **have** $\langle \text{finite } \{t. c_t \leq n \wedge c \ \nabla \ t\} \rangle$
unfolding *zeno-clock-def* **by** *simp*
moreover from *occurs-def* **have**
 $\langle \exists f. \forall t. t \leq n \wedge \text{occurs } t \ c \longrightarrow$
 $(\exists k. f \ k = t \wedge c_k \leq n \wedge c \ \nabla \ k) \rangle$ **by** *auto*
hence
 $\langle \exists f. \forall t \in \{t. t \leq n \wedge \text{occurs } t \ c\}. \exists k. f \ k = t \wedge k \in \{k. c_k \leq n \wedge c \ \nabla \ k\} \rangle$ **by** *simp*
hence $\langle \exists f. \{t. t \leq n \wedge \text{occurs } t \ c\} \subseteq \text{image } f \ \{k. c_k \leq n \wedge c \ \nabla \ k\} \rangle$
by *fastforce*
ultimately have $\langle \text{card } \{t. t \leq n \wedge \text{occurs } t \ c\} \leq \text{card } \{k. c_k \leq n \wedge c \ \nabla \ k\} \rangle$
using *surj-card-le* **by** *blast*
thus *?thesis*
unfolding *chrono-up-to-def* *chrono-dense-up-to-def* *occurs-def* **by** *simp*
qed

Number of event occurrences during a time window.

definition *occurrence-count* :: $\langle [\text{chronoclock}, \text{nat}, \text{nat}] \Rightarrow \text{nat} \rangle$

where $\langle \text{occurrence-count } c \ t_0 \ d \equiv \text{card } \{t. t_0 \leq t \wedge t < t_0 + d \wedge \text{occurs } t \ c\} \rangle$

The number of event occurrences is monotonous with regard to the window width.

lemma *occ-count-mono*:

assumes $\langle d' \geq d \rangle$
shows $\langle \text{occurrence-count } c \ t_0 \ d' \geq \text{occurrence-count } c \ t_0 \ d \rangle$
proof –
have $\text{finite}:\langle \text{finite } \{t::\text{nat}. t_0 \leq t \wedge t < t_0 + d' \wedge \text{occurs } t \ c\} \rangle$ **by** *simp*
from *assms* **have** *incl*:
 $\langle \{t::\text{nat}. t_0 \leq t \wedge t < t_0 + d \wedge \text{occurs } t \ c\} \subseteq \{t::\text{nat}. t_0 \leq t \wedge t < t_0 + d' \wedge \text{occurs } t \ c\} \rangle$ **by** *auto*
have $\langle \text{card } \{t::\text{nat}. t_0 \leq t \wedge t < t_0 + d \wedge \text{occurs } t \ c\}$
 $\leq \text{card } \{t::\text{nat}. t_0 \leq t \wedge t < t_0 + d' \wedge \text{occurs } t \ c\} \rangle$
using *card-mono*[*OF finite incl*] .
thus *?thesis* **using** *occurrence-count-def* **by** *simp*

qed

lemma *interval-diff*:

$\langle \{i::nat. m \leq i \wedge i \leq m+l \wedge P\ i\} = \{i::nat. i \leq m+l \wedge P\ i\} - \{i::nat. i < m \wedge P\ i\} \rangle$
by *auto*

corollary *interval-diff-le*:

$\langle \{i::nat. m+1 \leq i \wedge i < m+1+l+1 \wedge P\ i\} = \{i::nat. i \leq m+1+l \wedge P\ i\} - \{i::nat. i \leq m \wedge P\ i\} \rangle$
using *interval-diff*[*of* $\langle m+1 \rangle$ $\langle l \rangle$ *P*] *Suc-eq-plus1 less-Suc-eq-le* **by** *presburger*

lemma $\langle occurrence-count\ c\ (t_0+1)\ (d+1) = chrono-up-to\ c\ (t_0+d+1) - chrono-up-to\ c\ t_0 \rangle$

proof –

have *1*: $\langle finite\ \{t. t \leq t_0 \wedge occurs\ t\ c\} \rangle$ **by** *simp*
have *2*: $\langle \{t. t \leq t_0 \wedge occurs\ t\ c\} \subseteq \{t. t \leq t_0 + d + 1 \wedge occurs\ t\ c\} \rangle$ **by** *auto*
from *interval-diff-le*[*of* $\langle t_0 \rangle$ $\langle d \rangle$] *card-Diff-subset*[*OF* *1 2*]
show *?thesis* **unfolding** *occurrence-count-def chrono-up-to-def* **by** *simp*
qed

end

theory *LinguaFrancaTests*

imports *LinguaFrancaClocks*

begin

6 Tests

abbreviation $\langle c1::clock \equiv (\lambda t. t \geq 1 \wedge (t-1) \bmod 2 = 0) \rangle$

abbreviation $\langle c2::clock \equiv (\lambda t. t \geq 2 \wedge (t-2) \bmod 3 = 0) \rangle$

value $\langle c1\ 0 \rangle$

value $\langle c1\ 1 \rangle$

value $\langle c1\ 2 \rangle$

value $\langle c1\ 3 \rangle$

value $\langle c2\ 0 \rangle$

value $\langle c2\ 1 \rangle$

value $\langle c2\ 2 \rangle$

value $\langle c2\ 3 \rangle$

value $\langle c2\ 4 \rangle$

value $\langle c2\ 5 \rangle$

lemma $\langle kp-periodic\ 1\ 2\ c1 \rangle$

using *kp-periodic-def* **by** *simp*

lemma $\langle kp-periodic\ 2\ 3\ c2 \rangle$

```

using kp-periodic-def by simp

abbreviation ⟨c3 ≡ c1 ⊕ c2⟩

value ⟨map c1 [0,1,2,3,4,5,6,7,8,9,10]⟩
value ⟨map ($c1) [0,1,2,3,4,5,6,7,8,9,10,11]⟩
value ⟨map c2 [0,1,2,3,4,5,6,7,8,9,10]⟩
value ⟨map c3 [0,1,2,3,4,5,6,7,8,9,10]⟩

lemma interv-2:⟨{t::nat. t₀ ≤ t ∧ t < t₀ + 2 ∧ 1 ≤ t ∧ (t - 1) mod 2 = 0} =
{t. (t = t₀ ∨ t = t₀ + 1) ∧ 1 ≤ t ∧ (t - 1) mod 2 = 0}⟩
  by auto

lemma ⟨bounded 2 1 c1⟩
proof -
  have ⟨∀ t. tick-count c1 t 2 ≤ 1⟩
  proof -
    { fix t₀::nat
      have ⟨tick-count c1 t₀ 2 ≤ 1⟩
      proof (cases t₀)
        case 0
          hence ⟨tick-count c1 t₀ 2 = ticks-up-to c1 1⟩
          using tick-count-orig by (simp add: numeral-2-eq-2)
          also have ⟨... = card {t::nat. t ≤ 1 ∧ 1 ≤ t ∧ (t-1) mod 2 = 0}⟩
          unfolding ticks-up-to-def by simp
          also have ⟨... ≤ card {t::nat. t ≤ 1 ∧ 1 ≤ t}⟩
          by (metis (mono-tags, lifting) Collect-cong
              cancel-comm-monoid-add-class.diff-cancel le-antisym le-refl mod-0)
          also have ⟨... = card {t::nat. t = 1}⟩ by (metis le-antisym order-refl)
          also have ⟨... = 1⟩ by simp
          finally show ?thesis .
        case Suc
          next
          case (Suc nat)
          then show ?thesis
          proof (cases ⟨(t₀-1) mod 2 = 0⟩)
            case True
              with Suc have ⟨t₀ mod 2 ≠ 0⟩ by arith
              hence ⟨{t. (t = t₀ ∨ t = t₀ + 1) ∧ 1 ≤ t ∧ (t - 1) mod 2 = 0} =
{t₀}⟩
              using True by auto
              hence ⟨{t. t₀ ≤ t ∧ t < t₀ + 2 ∧ 1 ≤ t ∧ (t - 1) mod 2 = 0} = {t₀}⟩
              using interv-2 by simp
              thus ?thesis unfolding tick-count-def by simp
            case False
              next
              case (False)
              with Suc have ⟨t₀ mod 2 = 0⟩ by arith
              hence ⟨{t. (t = t₀ ∨ t = t₀ + 1) ∧ 1 ≤ t ∧ (t - 1) mod 2 = 0} =
{t₀+1}⟩
              by auto
          qed
        }
      }
    }
  }

```

```

    hence  $\langle \{t. t_0 \leq t \wedge t < t_0 + 2 \wedge 1 \leq t \wedge (t - 1) \bmod 2 = 0\} =$ 
 $\{t_0+1\} \rangle$ 
      using interv-2 by simp
      thus ?thesis unfolding tick-count-def by simp
    qed
  qed
}
thus ?thesis ..
qed
thus ?thesis using bounded-def by simp
qed
end

```