Clocks

Frédéric Boulanger

February 22, 2020

Contents


```
theory Clocks

imports Main

begin
```

1 Basic definitions

Time is represented as the natural numbers. A clock represents an event that may occur or not at any time. 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 = ‹nat ⇒ bool›
```

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_periodic :: ‹[nat, nat, clock] ⇒ bool›
  where ‹kp_periodic k p c ≡
    (p > 0) ∧ (∀n. c n = ((n ≥ k) ∧ ((n - k) mod p = 0)))›
```

A 1-periodic clock always ticks starting at its offset

```
lemma one_periodic_ticks:
  assumes ‹kp_periodic k 1 c›
      and ‹n ≥ k›
    shows ‹c n›
⟨proof⟩
```

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

```
definition ‹p_periodic p c ≡ (∃k. kp_periodic k p c)›

lemma p_periodic_intro[intro]:
  ‹kp_periodic k p c ⟹ p_periodic p c›
⟨proof⟩
```

No clock is 0-periodic.

```
lemma no_0_periodic:
  ‹¬p_periodic 0 c›
⟨proof⟩
```

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

```
definition ‹periodic c ≡ (∃p. p_periodic p c)›

lemma periodic_intro1[intro]:
  ‹p_periodic p c ⟹ periodic c›
⟨proof⟩

lemma periodic_intro2[intro]:
  ‹kp_periodic k p c ⟹ periodic c›
⟨proof⟩
```

1.2 Sporadic clocks

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

```
definition p_sporadic :: ‹[nat, clock] ⇒ bool›
  where ‹p_sporadic p c ≡ (∀t. c t ⟶ (∀t'. (t < t' ∧ t' ≤ t+p) ⟶ ¬(c t')))›
```

Any clock is 0-sporadic

```
lemma sporadic_0: ‹p_sporadic 0 c›
  ⟨proof⟩
```

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

```
definition ‹sporadic c ≡ (∃p > 0. p_sporadic p c)›

lemma sporadic_intro[intro]
  :‹⟦p_sporadic p c;p > 0⟧ ⟹ sporadic c›
⟨proof⟩
```

2 Properties of clocks

Some useful lemmas about modulo.

```
lemma mod_sporadic:
  assumes ‹((n::nat) mod p = 0)›
```

```
    shows ‹∀n'. (n < n' ∧ n' < n+p) ⟶ ¬(n' mod p = 0)›
⟨proof⟩
```

```
lemma mod_offset_sporadic:
  assumes ‹(n::nat) ≥ k›
      and ‹(n - k) mod p = 0›
    shows ‹∀n'. (n < n' ∧ n' < n+p) ⟶ ¬((n'-k) mod p = 0)›
⟨proof⟩
```

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

```
lemma periodic_suc_sporadic:
  assumes ‹p_periodic (Suc p) c›
    shows ‹p_sporadic p c›
⟨proof⟩
```

3 Merging clocks

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

```
definition merge :: ‹[clock, clock] ⇒ clock› (infix ‹⊕› 60)
  where ‹c1 ⊕ c2 ≡ λt. c1 t ∨ c2 t›
```

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

```
lemma merge_no_sporadic:
  ‹∃c c'. sporadic c ∧ sporadic c' ∧ ¬sporadic (c⊕c')›
⟨proof⟩
```

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

```
definition ticks_up_to :: ‹[clock, nat] ⇒ nat›
  where ‹ticks_up_to c n = card {t. t ≤ n ∧ c t}›
```

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

```
lemma ‹ticks_up_to c n ≤ Suc n›
⟨proof⟩
```

Counting event occurrences.

```
definition ‹count b n ≡ if b then Suc n else n›
```

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

```
lemma count_inc: ‹count b n ≤ Suc n›
  ⟨proof⟩
```

Alternative definition of the number of event occurrences using fold.

```
definition ticks_up_to_fold :: ‹[clock, nat] ⇒ nat›
  where ‹ticks_up_to_fold c n = fold count (map c [0..<Suc n]) 0›
```

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

```
fun ticks_up_to_fun :: ‹[clock, nat] ⇒ nat›
where
  ‹ticks_up_to_fun c 0 = count (c 0) 0›
| ‹ticks_up_to_fun c (Suc n) = count (c (Suc n)) (ticks_up_to_fun c n)›
```

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

```
lemma ticks_up_to_is_fun[code]: ‹ticks_up_to c n = ticks_up_to_fun c n›
⟨proof⟩
```

Number of event occurrences during an n instant window starting at t_0.

```
definition tick_count ::‹[clock, nat, nat] ⇒ nat›
  where ‹tick_count c t₀ n ≡ card {t. t₀ ≤ t ∧ t < t₀+n ∧ c t}›
```

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

```
lemma tick_count_mono:
  assumes ‹n' ≥ n›
    shows ‹tick_count c t₀ n' ≥ tick_count c t₀ n›
⟨proof⟩
```

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

```
lemma card_interval:‹card {t. t₀ ≤ t ∧ t < t₀+n} = n›
⟨proof⟩
```

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

```
lemma tick_count_bound: ‹tick_count c t₀ n ≤ n›
⟨proof⟩
```

No event occurrence occur in 0 instant.

```
lemma tick_count_0[code]: ‹tick_count c t₀ 0 = 0›
  ⟨proof⟩
```

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

```
lemma tick_count_orig[code]:
  ‹tick_count c 0 (Suc n) = ticks_up_to c n›
⟨proof⟩
```

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

```
lemma tick_count_diff[code]:
  ‹tick_count c (Suc t₀) n = (ticks_up_to c (t₀+n)) - (ticks_up_to c t₀)›
⟨proof⟩
```

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

```
lemma tick_count_merge:
  ‹tick_count (c⊕c') t₀ n  ≤ tick_count c t₀ n + tick_count c' t₀ n›
⟨proof⟩
```

4 Bounded clocks

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

```
definition bounded :: ‹[nat, nat, clock] ⇒ bool›
  where ‹bounded n m c ≡ ∀t. tick_count c t n ≤ m›
```

All clocks are (n,n)-bounded.

```
lemma bounded_n: ‹bounded n n c›
  ‹proof›
```

A sporadic clock is bounded.

```
lemma spor_bound:
  assumes ‹∀t::nat. c t ⟶ (∀t'. (t < t' ∧ t' ≤ t+n) ⟶ ¬(c t'))›
  shows ‹∀t::nat. card {t'. t ≤ t' ∧ t' ≤ t+n ∧ c t'} ≤ 1›
‹proof›
```

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

```
lemma spor_bounded:
  assumes ‹p_sporadic n c›
    shows ‹bounded (Suc n) 1 c›
‹proof›
```

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

```
lemma spor_bounded2:
  assumes ‹p_sporadic n c›
    shows ‹bounded (Suc (Suc n)) 2 c›
‹proof›
```

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›
  ‹proof›
```

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')›
‹proof›
```

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 (Suc n) 2 (c⊕c')›
‹proof›
```

5 Main theorems

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

```
lemma sporadic_bounded_min:
  assumes ‹p_sporadic n c›
      and ‹p_sporadic n' c'›
    shows ‹bounded (Suc (min n n')) 2 (c⊕c')›
⟨proof⟩
```

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

```
lemma sporadic_bounded_max:
  assumes ‹p_sporadic n c›
      and ‹p_sporadic n' c'›
    shows ‹bounded (max n n') (Suc n + Suc n') (c⊕c')›
⟨proof⟩
```

6 Tests

```
abbreviation ‹c1::clock ≡ (λt. t ≥ 1 ∧ (t-1) mod 2 = 0)›
abbreviation ‹c2::clock ≡ (λt. t ≥ 2 ∧ (t-2) mod 3 = 0)›

value ‹c1 0›
value ‹c1 1›
value ‹c1 2›
value ‹c1 3›

value ‹c2 0›
value ‹c2 1›
value ‹c2 2›
value ‹c2 3›
value ‹c2 4›
value ‹c2 5›

lemma ‹kp_periodic 1 2 c1›
  ⟨proof⟩

lemma ‹kp_periodic 2 3 c2›
  ⟨proof⟩

abbreviation ‹c3 ≡ c1 ⊕ c2›

value ‹map c1 [0,1,2,3,4,5,6,7,8,9,10]›
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}›
  ⟨proof⟩

lemma ‹bounded 2 1 c1›
⟨proof⟩

end
```