

Instances of Schneider's generalized protocol of clock synchronization.

Damian Barsotti

December 12, 2009

Abstract

Schneider [7] generalizes a number of protocols for Byzantine fault-tolerant clock synchronization and presents a uniform proof for their correctness. In Schneider's schema, each processor maintains a local clock by periodically adjusting each value to one computed by a convergence function applied to the readings of all the clocks. Then, correctness of an algorithm, i.e. that the readings of two clocks at any time are within a fixed bound of each other, is based upon some conditions on the convergence function. To prove that a particular clock synchronization algorithm is correct it suffices to show that the convergence function used by the algorithm meets Schneider's conditions.

Using the theorem prover Isabelle, we formalize the proofs that the convergence functions of two algorithms, namely, the Interactive Convergence Algorithm (ICA) of Lamport and Melliar-Smith [4] and the Fault-tolerant Midpoint algorithm of Lundelius-Lynch [5], meet Schneider's conditions. Furthermore, we experiment on handling some parts of the proofs with fully automatic tools like ICS[3] and CVC-lite[2].

These theories are part of a joint work with Alwen Tiu and Leonor P. Nieto [1]. In this work the correctness of Schneider schema was also verified using Isabelle (available at <http://afp.sourceforge.net/entries/GenClock.shtml>).

Contents

1	Interactive Convergence Algorithms (ICA)	2
1.1	Model of the system	2
1.1.1	Types in the formalization	2
1.1.2	Some constants	3
1.1.3	Convergence function	3
1.2	Translation Invariance property.	3
1.3	Precision Enhancement property	4
1.3.1	Auxiliary lemmas	4
1.3.2	Main theorem	11

1.4	Accuracy Preservation property	13
1.4.1	Main theorem	14
2	Fault-tolerant Midpoint algorithm	16
2.1	Model of the system	16
2.1.1	Types in the formalization	16
2.1.2	Some constants	17
2.1.3	Convergence function	17
2.2	Translation Invariance property.	18
2.2.1	Auxiliary lemmas	18
2.2.2	Main theorem	24
2.3	Precision Enhancement property	25
2.3.1	Auxiliary lemmas	25
2.3.2	Main theorem	33
2.4	Accuracy Preservation property	35
A	CVC-lite and ICS proofs	37
A.1	Lemma <code>abs_distrib_div</code>	37
A.2	Bound for Precision Enhancement property	38
A.3	Accuracy Preservation property	55

1 Interactive Convergence Algorithms (ICA)

theory *ICAInstance* **imports** *Complex-Main* **begin**

This algorithm is presented in [4].

A proof of the three properties can be found in [8].

1.1 Model of the system

The main ideas for the formalization of the system were obtained from [8].

1.1.1 Types in the formalization

The election of the basics types was based on [8]. There, the process are natural numbers and the real time and the clock readings are reals.

types

process = *nat*

time = *real* — real time

Clocktime = *real* — time of the clock readings (clock time)

1.1.2 Some constants

Here we define some parameters of the algorithm that we use: the number of process and the fix value that is used to discard the processes whose clocks differ more than this amount from the own one (see [8]). The defined constants must satisfy this axiom (if $np = 0$ we have a division by zero in the definition of the convergence function).

axiomatization

$np :: nat$ — Number of processes **and**
 $\Delta :: Clocktime$ — Fix value to discard processes **where**
constants-ax: $0 \leq \Delta \wedge np > 0$

We define also the set of process that the algorithm manage. This definition exist only for readability matters.

definition

$PR :: process\ set$ **where**
[simp]: $PR = \{..<np\}$

1.1.3 Convergence function

This functions is called “Egocentric Average” ([7])

In this algorithm each process has an array where it store the clocks readings from the others processes (including itself). We formalise that as a function from processes to clock time as [8].

First we define an auxiliary function. It takes a function of clock readings and two processes, and return de reading of the second process if the difference of the readings is grater than Δ , otherwise it returns the reading of the first one.

definition

$fiX :: [(process \Rightarrow Clocktime), process, process] \Rightarrow Clocktime$ **where**
 $fiX\ f\ p\ l = (if\ |f\ p - f\ l| \leq \Delta\ then\ (f\ l)\ else\ (f\ p))$

And finally the convergence function. This is defined with the builtin generalized summation over a set constructor of Isabelle. Also we had to use the overloaded *real* function to typecast de number np .

definition

$cfni :: [process, (process \Rightarrow Clocktime)] \Rightarrow Clocktime$ **where**
 $cfni\ p\ f = (\sum\ l \in \{..<np\}. fiX\ f\ p\ l) / (real\ np)$

1.2 Translation Invariance property.

We first need to prove this auxiliary lemma.

lemma *trans-inv'*:

```

( $\sum l \in \{..<np'\}$ .  $fiX (\lambda y. f y + x) p l$ ) =
  ( $\sum l \in \{..<np'\}$ .  $fiX f p l$ ) +  $real\ np' * x$ 
apply (induct-tac np')
apply (auto simp add: cfni-def fiX-def real-of-nat-Suc
  left-distrib real-diff-def lessThan-Suc)
done

```

```

theorem trans-inv:
 $\forall p f x . cfni\ p (\lambda y. f y + x) = cfni\ p f + x$ 
apply (auto simp add: cfni-def trans-inv' left-distrib
  divide-inverse constants-ax)
done

```

1.3 Precision Enhancement property

An informal proof of this theorem can be found in [8]

1.3.1 Auxiliary lemmas

```

lemma finitC:
   $C \subseteq PR \implies finite\ C$ 
proof –
  assume  $C \subseteq PR$ 
  thus ?thesis using finite-subset by auto
qed

```

```

lemma finitnpC:
   $finite\ (PR - C)$ 
proof –
  show ?thesis using finite-Diff by auto
qed

```

The next lemmas are about arithmetic properties of the generalized summation over a set constructor.

```

lemma sum-abs-triangle-ineq:
 $finite\ S \implies$ 
   $|\sum l \in S. (f::'a \Rightarrow 'b::ordered-idom)\ l| \leq (\sum l \in S. |f\ l|)$ 
  (is ...  $\implies$  ?P S)
proof(induct S rule: finite-induct)
  show ?P {} by simp
next
  fix  $F x$ 
  assume finit:  $finite\ F$  and
     $xnotinF: x \notin F$  and HI:  $|\setsum f F| \leq (\sum l \in F. |f\ l|)$ 
  hence  $|\setsum f (insert\ x\ F)| = |f\ x + \setsum f F|$ 
    by simp
  also
  have ...  $\leq |f\ x| + |\setsum f F|$ 
    using abs-triangle-ineq

```

```

    by auto
  also
  from HI have ... <= |f x| + (∑ l∈F. |f l|)
    by simp
  also
  from finit xnotinF have ... = (∑ l∈insert x F. |f l|)
    by simp
  finally
  show ?P (insert x F) .
qed

lemma sum-le:
  [[finite S ; ∀ r∈S. f r <= b]]
  ⇒
  (∑ l∈S. f l) <= real (card S) * b
  (is [[finite S ; ∀ r∈S. f r <= b]] ⇒ ?P S)
proof(induct S rule: finite-induct)
  show ?P {} by simp
next
  fix F x
  assume finit: finite F and xnotinF: x ∉ F and
    HI1: ∀ r∈F. f r ≤ b ⇒ setsum f F ≤ real (card F) * b
    and HI2: ∀ r∈insert x F. f r ≤ b
  from HI1 HI2 and finit and xnotinF
  have setsum f (insert x F) <= b + real (card F) * b
    by auto
  also
  have ... = real (Suc (card F)) * b
    by (simp add: left-distrib real-of-nat-Suc)
  also
  from finit xnotinF have ... = real (card (insert x F)) * b
    by simp
  finally
  show ?P (insert x F) .
qed

```

```

lemma sum-np-eq:
  assumes
    hC: C ⊆ PR
  shows
    (∑ l∈{..

```

ultimately
show *?thesis*
using *setsum-Un-disjoint*[**where** $A=C$ and $B=\{..<np\} - C$]
by *auto*
qed

lemma *abs-sum-np-ineq*:

assumes

$hC: C \subseteq PR$

shows

$|\sum_{l \in \{..<np\}} (f :: nat \Rightarrow real) l| \leq$
 $(\sum_{l \in C} |f l|) + (\sum_{l \in (\{..<np\} - C)} |f l|)$
(**is** *?abs-sum* \leq *?sumC* + *?sumnpC*)

proof –

from hC and *sum-np-eq*[**where** $f=f$]

have *?abs-sum* = $|\sum_{l \in C} f l| + |\sum_{l \in (\{..<np\} - C)} f l|$
(**is** *?abs-sum* = $|\sum_{l \in C} f l| + |\sum_{l \in (\{..<np\} - C)} f l|$)

by *simp*

also

from *abs-triangle-ineq*

have $\dots \leq |\sum_{l \in C} f l| + |\sum_{l \in (\{..<np\} - C)} f l|$.

also

have $\dots \leq ?sumC + ?sumnpC$

proof –

from hC *finitC* *sum-abs-triangle-ineq*

have $|\sum_{l \in C} f l| \leq ?sumC$ **by** *blast*

moreover

from *finitnpC* and

sum-abs-triangle-ineq[**where** $f=f$ and $S=PR-C$]

have $|\sum_{l \in (\{..<np\} - C)} f l| \leq ?sumnpC$

by *force*

ultimately

show *?thesis* **by** *arith*

qed

finally

show *?thesis* .

qed

The next lemmas are about the existence of bounds that are necessary in order to prove the Precision Enhancement theorem.

lemma *fiX-ubound*:

$fiX f p l \leq f p + \Delta$

proof(*cases* $|f p - f l| \leq \Delta$)

assume *asm*: $|f p - f l| \leq \Delta$

hence $fiX f p l = f l$ **by** (*simp add*: *fiX-def*)

also

from *asm* **have** $f l \leq f p + \Delta$ **by** *arith*

finally

show *?thesis* **by** *arith*

next
assume $asm: \neg |f p - f l| \leq \Delta$
hence $fiX f p l = f p$ **by** (*simp add: fiX-def*)
also
from asm **and** $constants-ax$ **have** $f p \leq f p + \Delta$ **by** *arith*
finally
show *?thesis* **by** *arith*
qed

lemma *fiX-lbound*:
 $f p - \Delta \leq fiX f p l$
proof(*cases* $|f p - f l| \leq \Delta$)
assume $asm: |f p - f l| \leq \Delta$
hence $fiX f p l = f l$ **by** (*simp add: fiX-def*)
also
from asm **have** $f p - \Delta \leq f l$ **by** *arith*
finally
show *?thesis* **by** *arith*

next
assume $asm: \neg |f p - f l| \leq \Delta$
with $constants-ax$ **have** $f p - \Delta \leq f p$ **by** *arith*
also
from asm **have** $f p = fiX f p l$ **by** (*simp add: fiX-def*)
finally
show *?thesis* **by** *arith*
qed

lemma *abs-fiX-bound*: $|fiX f p l - f p| \leq \Delta$
proof –

have $f p - \Delta \leq fiX f p l \wedge fiX f p l \leq f p + \Delta \longrightarrow$ *?thesis*
by *arith*
with *fiX-lbound* *fiX-ubound* **show** *?thesis* **by** *blast*
qed

lemma *abs-dif-fiX-bound*:

assumes
 $hbx: \forall l \in C. |f l - g l| \leq x$ **and**
 $hby: \forall l \in C. \forall m \in C. |f l - f m| \leq y$ **and**
 $hpC: p \in C$ **and**
 $hqC: q \in C$

shows
 $|fiX f p r - fiX g q r| \leq 2 * \Delta + x + y$

proof –
have $|fiX f p r - fiX g q r| =$
 $|fiX f p r - f p + f p - fiX g q r|$
by *auto*

also
have $\dots \leq |fiX f p r - f p| + |f p - fiX g q r|$

```

    by arith
  also
  from abs-fiX-bound
  have ... <= Δ + |f p - fiX g q r|
    by simp
  also
  have ... = Δ + |f p - g q + (g q - fiX g q r)|
    by (simp add:real-diff-def)
  also
  from abs-triangle-ineq[where a = f p - g q and
                        b = g q - fiX g q r]
  have ... <= Δ + |f p - g q| + |g q - fiX g q r|
    by simp
  also
  have ... = Δ + |f p - g q| + |fiX g q r - g q|
    by arith
  also
  from abs-fiX-bound
  have ... <= 2 * Δ + |f p - g q|
    by simp
  also
  have ... = 2 * Δ + |f p - f q + (f q - g q)|
    by (simp add:real-diff-def)
  also
  from abs-triangle-ineq[where a = f p - f q and
                        b = f q - g q]
  have ... <= 2 * Δ + |f p - f q| + |f q - g q|
    by simp
  finally
  show ?thesis using hbx hby hpC hqC
    by force
qed

```

lemma *abs-dif-fiX-bound-C-aux1*:

assumes

```

  hbx: ∀ l ∈ C. |f l - g l| <= x and
  hby1: ∀ l ∈ C. ∀ m ∈ C. |f l - f m| <= y and
  hby2: ∀ l ∈ C. ∀ m ∈ C. |g l - g m| <= y and
  hpC: p ∈ C and
  hqC: q ∈ C and
  hrC: r ∈ C

```

shows

```

|fiX f p r - fiX g q r| <= x + y

```

proof(cases |f p - f r| ≤ Δ)

case *True*

note *outer-IH = True*

show *?thesis*

proof(cases |g q - g r| ≤ Δ)

```

case True
show ?thesis
proof -
  from hpC and hby1 have 0 <= y by force
  with hrC and hbx have |f r - g r| <= x + y by auto
  with outer-IH and True show ?thesis
    by (auto simp add: fiX-def)
qed
next
case False
show ?thesis
proof -
  from outer-IH and False
  have |fiX f p r - fiX g q r| = |f r - g q|
    by (auto simp add: fiX-def)
  also
  have ... = |f r - f q + f q - g q| by simp
  also
  have ... <= |f r - f q| + |f q - g q|
    by arith
  also
  from hbx hby1 hpC hqC hrC have ... <= x + y by force
  finally
  show ?thesis .
qed
qed
next
case False
note outer-IH = False
show ?thesis
proof(cases |g q - g r| ≤ Δ)
  case True
  show ?thesis
  proof -
    from outer-IH and True
    have |fiX f p r - fiX g q r| = |f p - g r|
      by (auto simp add: fiX-def)
    also
    have ... = |f p - f r + f r - g r| by simp
    also
    from abs-triangle-ineq[where a = f p - f r and
      b = f r - g r]
    have ... <= |f p - f r| + |f r - g r|
      by (auto simp add: real-diff-def)
    also
    from hbx hby1 hpC hrC have ... <= x + y by force
    finally
    show ?thesis .
  qed
qed

```

```

next
  case False
  show ?thesis
  proof -
    from outer-IH and False
    have  $|fiX\ f\ p\ r - fiX\ g\ q\ r| = |f\ p - g\ q|$ 
      by (auto simp add: fiX-def)
    also
    have  $\dots = |f\ p - f\ q + f\ q - g\ q|$  by simp
    also
    from abs-triangle-ineq[where a = f p - f q and
      b = f q - g q]
    have  $\dots \leq |f\ p - f\ q| + |f\ q - g\ q|$ 
      by (auto simp add: real-diff-def)
    also
    from hbx hby1 hpC hqC have  $\dots \leq x + y$  by force
    finally
    show ?thesis .
  qed
qed
qed

```

lemma *abs-dif-fiX-bound-C-aux2*:

assumes

```

hbx:  $\forall l \in C. |f\ l - g\ l| \leq x$  and
hby1:  $\forall l \in C. \forall m \in C. |f\ l - f\ m| \leq y$  and
hby2:  $\forall l \in C. \forall m \in C. |g\ l - g\ m| \leq y$  and
hpC:  $p \in C$  and
hqC:  $q \in C$  and
hrC:  $r \in C$ 

```

shows

```

 $y \leq \Delta \longrightarrow |fiX\ f\ p\ r - fiX\ g\ q\ r| \leq x$ 

```

proof

```

assume hyd:  $y \leq \Delta$ 

```

```

show  $|fiX\ f\ p\ r - fiX\ g\ q\ r| \leq x$ 

```

proof–

```

from hpC and hrC and hby1 and hyd have  $|f\ p - f\ r| \leq \Delta$ 
  by force

```

moreover

```

from hqC and hrC and hby2 and hyd have  $|g\ q - g\ r| \leq \Delta$ 
  by force

```

moreover

```

from hrC and hbx have  $|f\ r - g\ r| \leq x$  by auto

```

ultimately

```

show ?thesis

```

```

  by (auto simp add: fiX-def)

```

qed

qed

lemma *abs-dif-fiX-bound-C*:
assumes
hbx: $\forall l \in C. |f l - g l| \leq x$ **and**
hby1: $\forall l \in C. \forall m \in C. |f l - f m| \leq y$ **and**
hby2: $\forall l \in C. \forall m \in C. |g l - g m| \leq y$ **and**
hpC: $p \in C$ **and**
hqC: $q \in C$ **and**
hrC: $r \in C$
shows
 $|fiX f p r - fiX g q r| \leq$
 $x + (if (y \leq \Delta) then 0 else y)$
proof (*cases* $y \leq \Delta$)
case *True*
with *abs-dif-fiX-bound-C-aux2* **and**
hbx **and** *hby1* **and** *hby2* **and** *hpC* **and** *hqC* **and** *hrC*
have $|fiX f p r - fiX g q r| \leq x$ **by** *blast*
with *True* **show** *?thesis* **by** *simp*
next
case *False*
with *abs-dif-fiX-bound-C-aux1* **and**
hbx **and** *hby1* **and** *hby2* **and** *hpC* **and** *hqC* **and** *hrC*
have $|fiX f p r - fiX g q r| \leq x + y$ **by** *blast*
with *False* **show** *?thesis* **by** *simp*
qed

1.3.2 Main theorem

theorem *prec-enh*:
assumes
hC: $C \subseteq PR$ **and**
hbx: $\forall l \in C. |f l - g l| \leq x$ **and**
hby1: $\forall l \in C. \forall m \in C. |f l - f m| \leq y$ **and**
hby2: $\forall l \in C. \forall m \in C. |g l - g m| \leq y$ **and**
hpC: $p \in C$ **and**
hqC: $q \in C$
shows $|cfni p f - cfni q g| \leq$
 $(real (card C) * (x + (if (y \leq \Delta) then 0 else y)) +$
 $real (card (\{..\<np\} - C)) * (2 * \Delta + x + y)) / real np$
 $(is |?dif-div-np| \leq ?B)$
proof –
have $|(\sum l \in \{..\<np\}. fiX f p l) -$
 $(\sum l \in \{..\<np\}. fiX g q l)| =$
 $|(\sum l \in \{..\<np\}. fiX f p l - fiX g q l)|$
 $(is |?dif| = |?dif'|)$
by (*simp add: real-diff-def setsum-addf setsum-negf*)
also
from *abs-sum-np-ineq hC*
have $\dots \leq$
 $(\sum l \in C. |fiX f p l - fiX g q l|) +$

```

    (∑ l∈({..np}-C). |fiX f p l - fiX g q l|)
  (is |?dif'| <= ?boundC' + ?boundnpC')
  by simp
also
have ... <=
  real (card C) * (x + (if (y <= Δ) then 0 else y)) +
  real (card ({..np}-C)) * (2 * Δ + x + y)
  (is ... <= ?boundC + ?boundnpC)
proof-
  have ?boundC' <= ?boundC
  proof -
    from abs-dif-fiX-bound-C and
      hbx and hby1 and hby2 and hpC and hqC
    have ∀ r ∈ C.
      |fiX f p r - fiX g q r| <= x +
        (if (y <= Δ) then 0 else y)
    by blast
    thus ?thesis using sum-le[where S=C] and finitC[OF hC]
    by force
  qed
moreover
have ?boundnpC' <= ?boundnpC
proof -
  from abs-dif-fiX-bound and
    hbx and hby1 and hpC and hqC
  have ∀ r ∈ ({..np}-C). |fiX f p r - fiX g q r| <= 2 * Δ + x + y
  by blast
  with finitnpC
  show ?thesis
  by (auto intro: sum-le)
qed
ultimately
show ?thesis by arith
qed
finally
have bound: |?dif| <= ?boundC + ?boundnpC .
thus ?thesis
proof-
  have ?dif-div-np = ?dif / real np
  by (auto simp add: cfni-def left-distrib
    divide-inverse real-diff-def)
  hence |cfni p f - cfni q g| = |?dif| / real np
  by force
  with bound show ?thesis
  by (auto simp add: cfni-def divide-inverse constants-ax)
qed
qed

```

1.4 Accuracy Preservation property

First, a simple lemma about an arithmetic propertie of the generalized summation over a set constructor.

lemma *sum-div-card*:

$$\begin{aligned} & (\sum l \in \{..<n::nat\}. f l) + q * real n = \\ & (\sum l \in \{..<n\}. f l + q) \\ & \text{(is } ?Sl n = ?Sr n) \end{aligned}$$

proof (*induct n*)

case 0 **thus** *?case* **by** *simp*

next

case (*Suc n*)

thus *?case*

by (*auto simp: real-of-nat-Suc right-distrib lessThan-Suc*)

qed

Next, some lemmas about bounds that are used in the proof of Accuracy Preservation

lemma *bound-aux-C*:

assumes

hby: $\forall l \in C. \forall m \in C. |f l - f m| \leq x$ **and**

hpC: $p \in C$ **and**

hqC: $q \in C$ **and**

hrC: $r \in C$

shows

$$|fiX f p r - f q| \leq x$$

proof (*cases* $|f p - f r| \leq \Delta$)

case *True*

then have $|fiX f p r - f q| = |f r - f q|$

by (*simp add: fiX-def*)

also

from *hby hqC hrC* **have** $\dots \leq x$ **by** *blast*

finally

show *?thesis* .

next

case *False*

then have $|fiX f p r - f q| = |f p - f q|$

by (*simp add: fiX-def*)

also

from *hby hpC hqC* **have** $\dots \leq x$ **by** *blast*

finally

show *?thesis* .

qed

lemma *bound-aux*:

assumes

hby: $\forall l \in C. \forall m \in C. |f l - f m| \leq x$ **and**

hpC: $p \in C$ **and**

hqC: $q \in C$

shows
 $|fiX f p r - f q| \leq x + \Delta$
proof (*cases* $|f p - f r| \leq \Delta$)
case *True*
then have $|fiX f p r - f q| = |f r - f q|$
by (*simp add: fiX-def*)
also
have $\dots = |(f r - f p) + (f p - f q)|$
by *arith*
also
have $\dots \leq |f p - f r| + |f p - f q|$
by *arith*
also
from *True* **have** $\dots \leq \Delta + |f p - f q|$ **by** *arith*
also
from *hby hpC hqC* **have** $\dots \leq \Delta + x$ **by** *simp*
finally
show *?thesis* **by** *simp*
next
case *False*
then have $|fiX f p r - f q| = |f p - f q|$
by (*simp add: fiX-def*)
also
from *hby hpC hqC* **have** $\dots \leq x$ **by** *blast*
finally
show *?thesis using constants-ax* **by** *arith*
qed

1.4.1 Main theorem

lemma *accur-pres:*

assumes

hC: $C \subseteq PR$ **and**

hby: $\forall l \in C. \forall m \in C. |f l - f m| \leq x$ **and**

hpC: $p \in C$ **and**

hqC: $q \in C$

shows $|cfni p f - f q| \leq$

$$\frac{\text{real}(\text{card } C) * x + \text{real}(\text{card}(\{..<np\} - C)) * (x + \Delta)}{\text{real } np}$$

$$(\text{is } ?abs1 \leq (?bC + ?bnpC) / \text{real } np)$$

proof –

from *abs-sum-np-ineq* **and** *hC* **have**

$$|\sum_{l \in \{..<np\}} fiX f p l - f q| \leq$$

$$(\sum_{l \in C} |fiX f p l - f q|) +$$

$$(\sum_{l \in (\{..<np\} - C)} |fiX f p l - f q|)$$

by *simp*

also

have

$$\dots \leq \text{real}(\text{card } C) * x +$$

$real (card (\{..<np\} - C)) * (x + \Delta)$

proof-
have $(\sum l \in C. |fiX f p l - f q|) \leq$
 $real (card C) * x$

proof-
from *bound-aux-C* **and**
hby **and** *hpC* **and** *hqC*
have $\forall r \in C.$
 $|fiX f p r - f q| \leq x$
by *blast*
thus *?thesis* **using** *sum-le*[**where** $S=C$] **and** *finitC*[*OF hC*]
by *force*

qed
moreover
have $(\sum l \in (\{..<np\} - C). |fiX f p l - f q|) \leq$
 $real (card (\{..<np\} - C)) * (x + \Delta)$

proof -
from *bound-aux* **and**
hby **and** *hpC* **and** *hqC*
have $\forall r \in (\{..<np\} - C).$
 $|fiX f p r - f q| \leq x + \Delta$
by *blast*
thus *?thesis* **using** *sum-le*[**where** $S=\{..<np\} - C$]
and *finitnpC*
by *force*

qed
ultimately
show *?thesis* **by** *arith*

qed
finally
have *bound*: $|\sum l \in \{..<np\}. fiX f p l - f q|$
 $\leq real (card C) * x + real (card (\{..<np\} - C)) * (x + \Delta)$

.

thus
?thesis

proof-
from *constants-ax* **have**
 $res: inverse (real np) * real np = 1$
by *auto*
have
 $(cfni p f - f q) * real np =$
 $(\sum l \in \{..<np\}. fiX f p l) * real np / real np - f q * real np$
by (*auto simp add: cfni-def real-diff-def left-distrib*)

also
have $\dots =$
 $(\sum l \in \{..<np\}. fiX f p l) - f q * real np$
by *simp*

also
from *sum-div-card*[**where** $f=fiX f p$ **and** $n=np$ **and** $q=- f q$]

```

have ... = ( $\sum l \in \{..<np\}. fiX f p l - f q$ )
  by (auto simp add: real-diff-def)
finally
have
  ( $cfni p f - f q$ ) * real np = ( $\sum l \in \{..<np\}. fiX f p l - f q$ )
  .
— cambia
hence
  ( $cfni p f - f q$ ) * real np / real np =
  ( $\sum l \in \{..<np\}. fiX f p l - f q$ ) / real np
  by auto
with constants-ax have
  ( $cfni p f - f q$ ) =
  ( $\sum l \in \{..<np\}. fiX f p l - f q$ ) / real np
by simp
hence |  $cfni p f - f q$  | =
  |( $\sum l \in \{..<np\}. fiX f p l - f q$ ) / real np |
  by simp
also have
  ... = |( $\sum l \in \{..<np\}. fiX f p l - f q$ )| / real np
  by auto
finally have |  $cfni p f - f q$  | =
  |( $\sum l \in \{..<np\}. fiX f p l - f q$ )| / real np
  .
with bound show ?thesis
  by (auto simp add: cfni-def divide-inverse constants-ax)
qed
qed

end

```

2 Fault-tolerant Midpoint algorithm

theory *LynchInstance* **imports** *Complex-Main* **begin**

This algorithm is presented in [5].

2.1 Model of the system

The main ideas for the formalization of the system were obtained from [8].

2.1.1 Types in the formalization

The election of the basics types was based on [8]. There, the process are natural numbers and the real time and the clock readings are reals.

types

$process = nat$
 $time = real$ — real time
 $Clocktime = real$ — time of the clock readings (clock time)

2.1.2 Some constants

Here we define some parameters of the algorithm that we use: the number of process and the number of lowest and highest readed values that the algorithm discards. The defined constants must satisfy this axiom. If not, the algorithm cannot obtain the maximum and minimum value, because it will have discarded all the values.

axiomatization

$np :: nat$ — Number of processes **and**
 $khl :: nat$ — Number of lowest and highest values **where**
constants-ax: $2 * khl < np$

We define also the set of process that the algorithm manage. This definition exist only for readability matters.

definition

$PR :: process\ set$ **where**
[simp]: $PR = \{..<np\}$

2.1.3 Convergence function

This functions is called “Fault-tolerant Midpoint” ([7])

In this algorithm each process has an array where it store the clocks readings from the others processes (including itself). We formalise that as a function from processes to clock time as [8].

First we define two functions. They take a function of clock readings and a set of processes and they return a set of khl processes which has the greater (smaller) clock readings. They were defined with the Hilbert’s ε -operator (the indefinite description operator *SOME* in Isabelle) because in this way the formalization is not fixed to a particular election of the processes’s readings to discards and then the modelization is more general.

definition

$kmax :: (process \Rightarrow Clocktime) \Rightarrow process\ set \Rightarrow process\ set$ **where**
 $kmax\ f\ P = (SOME\ S. S \subseteq P \wedge card\ S = khl \wedge$
 $(\forall\ i \in S. \forall\ j \in (P - S). f\ j \leq f\ i))$

definition

$kmin :: (process \Rightarrow Clocktime) \Rightarrow process\ set \Rightarrow process\ set$ **where**
 $kmin\ f\ P = (SOME\ S. S \subseteq P \wedge card\ S = khl \wedge$
 $(\forall\ i \in S. \forall\ j \in (P - S). f\ i \leq f\ j))$

With the previous functions we define a new one *reduce*¹. This takes a function of clock readings and a set of processes and returns the set of readings of the not discarded processes. In order to define this function we use the image operator (*op* ‘) of Isabelle.

definition

```
reduce :: (process ⇒ Clocktime) ⇒ process set ⇒ Clocktime set where
reduce f P = f ‘ (P - (kmax f P ∪ kmin f P))
```

And finally the convergence function. This is defined with the builtin *Max* and *Min* functions of Isabelle.

definition

```
cfnl :: process ⇒ (process ⇒ Clocktime) ⇒ Clocktime where
cfnl p f = (Max (reduce f PR) + Min (reduce f PR)) / 2
```

2.2 Translation Invariance property.

2.2.1 Auxiliary lemmas

These lemmas prove the existence of the maximum and minimum of the image of a set, if the set is finite and not empty.

lemma *ex-Maxf*:

```
fixes S and f :: 'a ⇒ ('b::linorder)
```

```
  assumes fin: finite S
```

```
  shows S ≠ {} ==> ∃ m ∈ S. ∀ s ∈ S. f s ≤ f m
```

```
using fin
```

```
proof (induct)
```

```
  case empty thus ?case by simp
```

```
next
```

```
  case (insert x S)
```

```
  show ?case
```

```
  proof (cases)
```

```
    assume S = {} thus ?thesis by simp
```

```
  next
```

```
    assume nonempty: S ≠ {}
```

```
    then obtain m where m: m ∈ S ∧ ∀ s ∈ S. f s ≤ f m
```

```
      using insert by blast
```

```
    show ?thesis
```

```
    proof (cases)
```

```
      assume f x ≤ f m thus ?thesis using m by blast
```

```
    next
```

```
      assume ~ f x ≤ f m thus ?thesis using m
```

```
        by(simp add:linorder-not-le order-less-le)
```

```
        (blast intro: order-trans)
```

```
    qed
```

```
  qed
```

```
qed
```

¹The name of this function was taken from [5].

```

lemma ex-Minf:
fixes  $S$  and  $f :: 'a \Rightarrow ('b::linorder)$ 
  assumes  $fin$ : finite  $S$ 
  shows  $S \neq \{\}$   $\implies \exists m \in S. \forall s \in S. f\ m \leq f\ s$ 
using  $fin$ 
proof (induct)
  case empty thus ?case by simp
next
  case (insert  $x\ S$ )
  show ?case
  proof (cases)
    assume  $S = \{\}$  thus ?thesis by simp
  next
    assume nonempty:  $S \neq \{\}$ 
    then obtain  $m$  where  $m: m \in S \ \forall s \in S. f\ m \leq f\ s$ 
      using insert by blast
    show ?thesis
    proof (cases)
      assume  $f\ m \leq f\ x$  thus ?thesis using  $m$  by blast
    next
      assume  $\sim f\ m \leq f\ x$  thus ?thesis using  $m$ 
        by(simp add:linorder-not-le order-less-le)
          (blast intro: order-trans)
    qed
  qed
qed

```

This trivial lemma is needed by the next two.

```

lemma khl-bound:  $khl < np$ 
  using constants-ax by arith

```

The next two lemmas prove that `de` functions `kmin` and `kmax` return some values that satisfy their definition. This is not trivial because we need to prove the existence of these values, according to the rule of the Hilbert's operator. We will need this lemma many times because it is the only thing that we know about these functions.

```

lemma kmax-prop:
fixes  $f :: nat \Rightarrow Clocktime$ 
  shows
 $(kmax\ f\ PR) \subseteq PR \wedge card\ (kmax\ f\ PR) = khl \wedge$ 
 $(\forall i \in (kmax\ f\ PR). \forall j \in PR - (kmax\ f\ PR). f\ j \leq f\ i)$ 
proof –
  have  $khl \leq np \longrightarrow$ 
     $(\exists S. S \subseteq PR \wedge card\ S = khl \wedge (\forall i \in S. \forall j \in PR - S. f\ j \leq f\ i))$ 
    (is  $khl \leq np \longrightarrow ?P\ khl$ )
  proof(induct  $khl$ )
    have ?P 0 by force
    thus  $0 \leq np \longrightarrow ?P\ 0 ..$ 

```

next
fix n
assume $asm: n \leq np \longrightarrow ?P\ n$
show $Suc\ n \leq np \longrightarrow ?P\ (Suc\ n)$
proof
assume $asm2: Suc\ n \leq np$
with asm **have** $?P\ n$ **by** $simp$
then obtain S **where**
 $SinPR : S \subseteq PR$ **and**
 $cardS: card\ S = n$ **and**
 $HI: (\forall i \in S. \forall j \in PR - S. f\ j \leq f\ i)$
by $blast$
let $?e = SOME\ i. i \in PR - S \wedge$
 $(\forall j \in PR - S. f\ j \leq f\ i)$
let $?S = insert\ ?e\ S$
have $\exists i. i \in PR - S \wedge (\forall j \in PR - S. f\ j \leq f\ i)$
proof-
from $SinPR$ **and** $finite-subset$
have $finite\ (PR - S)$
by $auto$
moreover
from $cardS$ **and** $asm2\ SinPR$
have $S \subset PR$ **by** $auto$
hence $PR - S \neq \{\}$ **by** $auto$
ultimately
show $?thesis$ **using** $ex-Maxf$ **by** $blast$
qed
hence
 $ePRS: ?e \in PR - S$ **and** $maxH: (\forall j \in PR - S. f\ j \leq f\ ?e)$
by $(auto\ dest!: someI-ex)$
from $maxH$ **and** HI
have $(\forall i \in ?S. \forall j \in PR - ?S. f\ j \leq f\ i)$
by $blast$
moreover
from $SinPR$ **and** $finite-subset$
 $cardS$ **and** $ePRS$
have $card\ ?S = Suc\ n$
by $(auto\ dest: card-insert-disjoint)$
moreover
have $?S \subseteq PR$ **using** $SinPR$ **and** $ePRS$ **by** $auto$
ultimately
show $?P\ (Suc\ n)$ **by** $blast$
qed
qed
hence $?P\ khl$ **using** $khl-bound$ **by** $auto$
then obtain S **where**
 $S \leq PR \wedge card\ S = khl \wedge (\forall i \in S. \forall j \in PR - S. f\ j \leq f\ i) ..$
thus $?thesis$ **by** $(unfold\ kmax-def)$
 $(rule\ someI\ [where\ P = \lambda S. S \subseteq PR \wedge card\ S = khl \wedge (\forall i \in S. \forall j \in PR - S.$

$f j \leq f i$)])
qed

lemma *kmin-prop*:

fixes $f :: nat \Rightarrow Clocktime$

shows

$(kmin\ f\ PR) \subseteq PR \wedge card\ (kmin\ f\ PR) = khl \wedge$
 $(\forall i \in (kmin\ f\ PR). \forall j \in PR - (kmin\ f\ PR). f\ i \leq f\ j)$

proof–

have $khl \leq np \longrightarrow$

$(\exists S. S \subseteq PR \wedge card\ S = khl \wedge (\forall i \in S. \forall j \in PR - S. f\ i \leq f\ j))$

(is $khl \leq np \longrightarrow ?P\ khl$)

proof(*induct khl*)

have $?P\ 0$ **by** *force*

thus $0 \leq np \longrightarrow ?P\ 0$ **..**

next

fix n

assume $asm: n \leq np \longrightarrow ?P\ n$

show $Suc\ n \leq np \longrightarrow ?P\ (Suc\ n)$

proof

assume $asm2: Suc\ n \leq np$

with asm **have** $?P\ n$ **by** *simp*

then obtain S **where**

$SinPR : S \subseteq PR$ **and**

$cardS: card\ S = n$ **and**

$HI: (\forall i \in S. \forall j \in PR - S. f\ i \leq f\ j)$

by *blast*

let $?e = SOME\ i. i \in PR - S \wedge$

$(\forall j \in PR - S. f\ i \leq f\ j)$

let $?S = insert\ ?e\ S$

have $\exists i. i \in PR - S \wedge (\forall j \in PR - S. f\ i \leq f\ j)$

proof–

from $SinPR$ **and** *finite-subset*

have *finite* $(PR - S)$

by *auto*

moreover

from $cardS$ **and** $asm2\ SinPR$

have $S \subset PR$ **by** *auto*

hence $PR - S \neq \{\}$ **by** *auto*

ultimately

show $?thesis$ **using** *ex-Minf* **by** *blast*

qed

hence

$ePRS: ?e \in PR - S$ **and** $minH: (\forall j \in PR - S. f\ ?e \leq f\ j)$

by (*auto dest!:* *someI-ex*)

from $minH$ **and** HI

have $(\forall i \in ?S. \forall j \in PR - ?S. f\ i \leq f\ j)$

by *blast*

moreover

```

from SinPR and finite-subset and
  cardS and ePRS
have card ?S = Suc n
  by (auto dest: card-insert-disjoint)
moreover
have ?S ⊆ PR using SinPR and ePRS by auto
ultimately
show ?P (Suc n) by blast
qed
qed
hence ?P khl using khl-bound by auto
then obtain S where
  S ≤ PR ∧ card S = khl ∧ (∀ i ∈ S. ∀ j ∈ PR - S. f i ≤ f j) ..
  thus ?thesis by (unfold kmin-def)
  (rule someI [where P = λS. S ⊆ PR ∧ card S = khl ∧ (∀ i ∈ S. ∀ j ∈ PR - S.
f i ≤ f j)])
qed

```

The next two lemmas are trivial from the previous ones

```

lemma finite-kmax:
finite (kmax f PR)
proof –
  have finite PR by auto
  with kmax-prop and finite-subset show ?thesis
  by blast
qed

```

```

lemma finite-kmin:
finite (kmin f PR)
proof –
  have finite PR by auto
  with kmin-prop and finite-subset show ?thesis
  by blast
qed

```

This lemma is necessary because the definition of the convergence function use the builtin Max and Min.

```

lemma reduce-not-empty:
reduce f PR ≠ {}
proof –
  from constants-ax have
    0 < (np - 2 * khl) by arith
  also
  {
    from kmax-prop kmin-prop
    have card (kmax f PR) = khl ∧ card (kmin f PR) = khl
    by blast
    moreover
    from finite-kmax and finite-kmin card-Un-Int [THEN sym]
  }

```

```

have card (kmax f PR  $\cup$  kmin f PR) +
  card (kmax f PR  $\cap$  kmin f PR) =
  card (kmax f PR) + card (kmin f PR)
  by auto
ultimately
have card (kmax f PR  $\cup$  kmin f PR)  $\leq$  2 * khl
  by auto
}
hence
...  $\leq$  card PR - card (kmax f PR  $\cup$  kmin f PR)
by simp
also
{
  from kmax-prop and kmin-prop have
  (kmax f PR  $\cup$  kmin f PR)  $\subseteq$  PR by blast
}
hence
... = card (PR - (kmax f PR  $\cup$  kmin f PR))
apply (intro card-Diff-subset[THEN sym])
apply (rule finite-subset)
by auto

finally
have 0 < card (PR - (kmax f PR  $\cup$  kmin f PR)) .
hence (PR - (kmax f PR  $\cup$  kmin f PR))  $\neq$  {}
  by (intro notI, simp only: card-0-eq, simp)
thus ?thesis
  by (auto simp add: reduce-def)
qed

```

The next three are the main lemmas necessary for prove the Translation Invariance property.

```

lemma reduce-shift:
fixes f :: nat  $\Rightarrow$  Clocktime
  shows
  f ' (PR - (kmax f PR  $\cup$  kmin f PR)) =
    f ' (PR - (kmax ( $\lambda$  p. f p + c) PR  $\cup$  kmin ( $\lambda$  p. f p + c) PR))
apply (unfold kmin-def kmax-def)
by simp

```

```

lemma max-shift:
fixes f :: nat  $\Rightarrow$  Clocktime and S
assumes notEmpFin: S  $\neq$  {} finite S
shows
Max (f'S) + x = Max ( ( $\lambda$  p. f p + x) ' S)
proof -
  from notEmpFin have f'S  $\neq$  {} and ( $\lambda$  p. f p + x) ' S  $\neq$  {}
  by auto
  with notEmpFin have

```

$Max (f'S) \in f' S$ $Max ((\lambda p. f p + x)'S) \in (\lambda p. f p + x)' S$
 $(\forall fs \in (f'S). fs \leq Max (f'S))$
 $(\forall fs \in ((\lambda p. f p + x)'S). fs \leq Max ((\lambda p. f p + x)'S))$
 by *auto*
 thus *?thesis by force*
qed

lemma *min-shift*:

fixes $f :: nat \Rightarrow Clocktime$ **and** S

assumes $notEmpFin: S \neq \{\}$ *finite* S

shows

$Min (f'S) + x = Min ((\lambda p. f p + x)' S)$

proof –

from $notEmpFin$ **have** $f'S \neq \{\}$ **and** $(\lambda p. f p + x)' S \neq \{\}$

by *auto*

with $notEmpFin$ **have**

$Min (f'S) \in f' S$ $Min ((\lambda p. f p + x)'S) \in (\lambda p. f p + x)' S$

$(\forall fs \in (f'S). Min (f'S) \leq fs)$

$(\forall fs \in ((\lambda p. f p + x)'S). Min ((\lambda p. f p + x)'S) \leq fs)$

by *auto*

thus *?thesis by force*

qed

2.2.2 Main theorem

theorem *trans-inv*:

fixes $f :: nat \Rightarrow Clocktime$

shows

$cfnl p f + x = cfnl p (\lambda p. f p + x)$

proof –

have $cfnl p (\lambda p. f p + x) =$

$(Max (reduce (\lambda p. f p + x) PR) + Min (reduce (\lambda p. f p + x) PR)) / 2$

by (*unfold cfnl-def, simp*)

also

have ... =

$(Max ((\lambda p. f p + x)'$
 $(PR - (kmax (\lambda p. f p + x) PR \cup kmin (\lambda p. f p + x) PR))) +$

$Min ((\lambda p. f p + x)'$
 $(PR - (kmax (\lambda p. f p + x) PR \cup kmin (\lambda p. f p + x) PR)))) / 2$

by (*unfold reduce-def, simp*)

also

have

... =

$(Max (f'$
 $(PR - (kmax (\lambda p. f p + x) PR \cup kmin (\lambda p. f p + x) PR))) + x +$

$Min (f'$
 $(PR - (kmax (\lambda p. f p + x) PR \cup kmin (\lambda p. f p + x) PR))) + x) / 2$

proof –

have *finite* $(PR - (kmax (\lambda p. f p + x) PR \cup kmin (\lambda p. f p + x) PR))$

by *auto*
moreover
from *reduce-not-empty* **have**
 $PR - (kmax (\lambda p. f p + x) PR \cup kmin (\lambda p. f p + x) PR) \neq \{\}$
by (*auto simp add: reduce-def*)
ultimately
have
 $Max ((\lambda p. f p + x) \text{'}$
 $(PR - (kmax (\lambda p. f p + x) PR \cup kmin (\lambda p. f p + x) PR)))$
 $=$
 $Max (f \text{'}$
 $(PR - (kmax (\lambda p. f p + x) PR \cup kmin (\lambda p. f p + x) PR))) + x$
and
 $Min ((\lambda p. f p + x) \text{'}$
 $(PR - (kmax (\lambda p. f p + x) PR \cup kmin (\lambda p. f p + x) PR)))$
 $=$
 $Min (f \text{'}$
 $(PR - (kmax (\lambda p. f p + x) PR \cup kmin (\lambda p. f p + x) PR))) + x$
using *max-shift* **and** *min-shift*
by *auto*
thus *?thesis* **by** *auto*
qed
also
from *reduce-shift*
have
 $\dots =$
 $(Max (f \text{'}$
 $(PR - (kmax f PR \cup kmin f PR))) + x +$
 $Min (f \text{'}$
 $(PR - (kmax f PR \cup kmin f PR))) + x) / 2$
by *auto*
also
have $\dots = ((Max (reduce f PR) + x) + (Min (reduce f PR) + x)) / 2$
by (*auto simp add: reduce-def*)
also
have $\dots = (Max (reduce f PR) + Min (reduce f PR)) / 2 + x$
by *auto*
finally
show *?thesis* **by** (*auto simp add: cfnl-def*)
qed

2.3 Precision Enhancement property

An informal proof of this theorem can be found in [6]

2.3.1 Auxiliary lemmas

This first lemma is most important for prove the property. This is a consequence of the *card-Un-Int* lemma

lemma *pigeonhole*:
assumes
finitA: *finite A* **and**
Bss: $B \subseteq A$ **and** *Css*: $C \subseteq A$ **and**
cardH: $\text{card } A + k \leq \text{card } B + \text{card } C$
shows $k \leq \text{card } (B \cap C)$
proof –
from *Bss* *Css* **have** $B \cup C \subseteq A$ **by** *blast*
with *finitA* **have** $\text{card } (B \cup C) \leq \text{card } A$
by (*simp add: card-mono*)
with *cardH* **have**
h: $k \leq \text{card } B + \text{card } C - \text{card } (B \cup C)$
by *arith*
from *finitA* *Bss* *Css* **and** *finite-subset*
have $\text{finite } B \wedge \text{finite } C$ **by** *auto*
thus *?thesis*
using *card-Un-Int* **and** *h* **by** *force*
qed

This lemma is a trivial consequence of the previous one. With only this lemma we can prove the Precision Enhancement property with the bound $\pi(x, y) = x + y$. But this bound not satisfy the property

$$\pi(2\Lambda + 2\beta\rho, \delta_S + 2\rho(r_{max} + \beta) + 2\Lambda) \leq \delta_S$$

that is used in [8] for prove the Schneider's schema.

lemma *subsets-int*:
assumes
finitA: *finite A* **and**
Bss: $B \subseteq A$ **and** *Css*: $C \subseteq A$ **and**
cardH: $\text{card } A < \text{card } B + \text{card } C$
shows
 $B \cap C \neq \{\}$
proof –
from *finitA* *Bss* *Css* *cardH*
have $1 \leq \text{card } (B \cap C)$
by (*auto intro!: pigeonhole*)
thus *?thesis* **by** *auto*
qed

This lemma is true because *reduce f PR* is the image of $PR - (kmax f PR \cup kmin f PR)$ by the function *f*.

lemma *exist-reduce*:
 $\forall c \in \text{reduce } f PR. \exists i \in PR - (kmax f PR \cup kmin f PR). f i = c$
proof
fix *c* **assume** *asm*: $c \in \text{reduce } f PR$
thus $\exists i \in PR - (kmax f PR \cup kmin f PR). f i = c$
by (*auto simp add: reduce-def kmax-def kmin-def*)
qed

The next three lemmas are consequence of the definition of *reduce*, *kmax* and *kmin*

lemma *finite-reduce*:

finite (*reduce* *f* *PR*)

proof(*unfold reduce-def*)

show *finite* (*f* ‘ (*PR* – (*kmax* *f* *PR* ∪ *kmin* *f* *PR*)))

by *auto*

qed

lemma *kmax-ge*:

$\forall i \in (kmax\ f\ PR). \forall r \in (reduce\ f\ PR). r \leq f\ i$

proof

fix *i* **assume** *asm*: $i \in kmax\ f\ PR$

show $\forall r \in reduce\ f\ PR. r \leq f\ i$

proof

fix *r* **assume** *asm2*: $r \in reduce\ f\ PR$

show $r \leq f\ i$

proof–

from *asm2* **and** *exist-reduce* **have**

$\exists j \in PR - (kmax\ f\ PR \cup kmin\ f\ PR). f\ j = r$ **by** *blast*

then obtain *j*

where *fjr*: $j \in PR - (kmax\ f\ PR \cup kmin\ f\ PR) \wedge f\ j = r$

by *blast*

hence $j \in (PR - kmax\ f\ PR)$

by *blast*

from *this fjr asm*

show *?thesis* **using** *kmax-prop*

by *auto*

qed

qed

qed

lemma *kmin-le*:

$\forall i \in (kmin\ f\ PR). \forall r \in (reduce\ f\ PR). f\ i \leq r$

proof

fix *i* **assume** *asm*: $i \in kmin\ f\ PR$

show $\forall r \in reduce\ f\ PR. f\ i \leq r$

proof

fix *r* **assume** *asm2*: $r \in reduce\ f\ PR$

show $f\ i \leq r$

proof–

from *asm2* **and** *exist-reduce* **have**

$\exists j \in PR - (kmax\ f\ PR \cup kmin\ f\ PR). f\ j = r$ **by** *blast*

then obtain *j*

where *fjr*: $j \in PR - (kmax\ f\ PR \cup kmin\ f\ PR) \wedge f\ j = r$

by *blast*

hence $j \in (PR - kmin\ f\ PR)$

by *blast*

from *this fjr asm*

```

    show ?thesis using kmin-prop
    by auto
  qed
qed
qed

```

The next lemma is used for prove the Precision Enhancement property. This has been proved in ICS. The proof is in the appendix A.1. This cannot be prove by a simple *arith* or *auto* tactic.

This lemma is true also with $0 \leq c$!!

```

lemma abs-distrib-div:
  0 < (c::real)  $\implies$  |a / c - b / c| = |a - b| / c
proof-
  assume ch: 0 < c
  {
    fix d :: real
    assume dh: 0 <= d
    have a * d - b * d = (a - b) * d
      by (auto simp add: left-distrib real-diff-def)
    hence |a * d - b * d| = |(a - b) * d|
      by simp
    also with dh have
      ... = |a - b| * d
      by (simp add: abs-mult)
    finally
      have |a * d - b * d| = |a - b| * d
        .
  }
  with ch and divide-inverse show ?thesis
  by (auto simp add: divide-inverse)
qed

```

The next three lemmas are about the existence of bounds of the values *Max* (*reduce f PR*) and *Min* (*reduce f PR*). These are used in the proof of the main property.

```

lemma uboundmax:
  assumes
    hC: C  $\subseteq$  PR and
    hCk: np <= card C + khl
  shows
     $\exists i \in C. \text{Max} (\text{reduce } f \text{ PR}) \leq f i$ 
proof-
  from reduce-not-empty and finite-reduce
  have maxrinr: Max (reduce f PR)  $\in$  reduce f PR
    by simp
  with exist-reduce

```

have $\exists i \in PR - (kmax\ f\ PR \cup kmin\ f\ PR). f\ i = Max\ (reduce\ f\ PR)$
by *simp*
then obtain *pmax* **where**
pmax-in-reduc: $pmax \in PR - (kmax\ f\ PR \cup kmin\ f\ PR)$ **and**
fpmax-ismax: $f\ pmax = Max\ (reduce\ f\ PR)$..
hence $C \cap insert\ pmax\ (kmax\ f\ PR) \neq \{\}$
proof–
from *kmax-prop* **and** *pmax-in-reduc*
and *finite-kmax* **and** *hCk* **have**
 $card\ PR < card\ C + card\ (insert\ pmax\ (kmax\ f\ PR))$
by *simp*
moreover
from *pmax-in-reduc* **and** *kmax-prop*
have $insert\ pmax\ (kmax\ f\ PR) \subseteq PR$ **by** *blast*
moreover
note *hC*
ultimately
show *?thesis*
using *subsets-int[of PR C insert pmax (kmax f PR)]*
by *simp*
qed
hence *res*: $\exists i \in C. i = pmax \vee i \in kmax\ f\ PR$ **by** *blast*
then obtain *i* **where**
iinC: $i \in C$ **and** *altern*: $i = pmax \vee i \in kmax\ f\ PR$..
thus *?thesis*
proof(*cases i=pmax*)
case *True*
with *iinC fpmax-ismax* **show** *?thesis* **by** *force*
next
case *False*
with *altern maxrinv fpmax-ismax kmax-ge*
have $f\ pmax \leq f\ i$ **by** *simp*
with *iinC fpmax-ismax* **show** *?thesis* **by** *auto*
qed
qed

lemma *lboundmin*:
assumes
hC: $C \subseteq PR$ **and**
hCk: $np \leq card\ C + khl$
shows
 $\exists i \in C. f\ i \leq Min\ (reduce\ f\ PR)$
proof–
from *reduce-not-empty* **and** *finite-reduce*
have *minrinv*: $Min\ (reduce\ f\ PR) \in reduce\ f\ PR$
by *simp*
with *exist-reduce*
have $\exists i \in PR - (kmax\ f\ PR \cup kmin\ f\ PR). f\ i = Min\ (reduce\ f\ PR)$
by *simp*

then obtain $pmin$ where
pmin-in-reduc: $pmin \in PR - (kmax\ f\ PR \cup kmin\ f\ PR)$ **and**
fpmin-ismin: $f\ pmin = Min\ (reduce\ f\ PR)$..
hence $C \cap insert\ pmin\ (kmin\ f\ PR) \neq \{\}$
proof–
from *kmin-prop* **and** *pmin-in-reduc*
and *finite-kmin* **and** *hCk* **have**
 $card\ PR < card\ C + card\ (insert\ pmin\ (kmin\ f\ PR))$
by *simp*
moreover
from *pmin-in-reduc* **and** *kmin-prop*
have $insert\ pmin\ (kmin\ f\ PR) \subseteq PR$ **by** *blast*
moreover
note *hC*
ultimately
show *?thesis*
using *subsets-int[of PR C insert pmin (kmin f PR)]*
by *simp*
qed
hence $res: \exists\ i \in C. i = pmin \vee i \in kmin\ f\ PR$ **by** *blast*
then obtain i **where**
iinC: $i \in C$ **and** *altern*: $i = pmin \vee i \in kmin\ f\ PR$..
thus *?thesis*
proof(*cases i=pmin*)
case *True*
with *iinC fpmin-ismin* **show** *?thesis* **by** *force*
next
case *False*
with *altern minrnr fpmin-ismin kmin-le*
have $f\ i \leq f\ pmin$ **by** *simp*
with *iinC fpmin-ismin* **show** *?thesis* **by** *auto*
qed
qed

lemma *same-bound*:
assumes
hC: $C \subseteq PR$ **and**
hCk: $np \leq card\ C + khl$ **and**
hnk: $3 * khl < np$
shows
 $\exists\ i \in C. Min\ (reduce\ f\ PR) \leq f\ i \wedge g\ i \leq Max\ (reduce\ g\ PR)$
proof–
have *b1*: $khl + 1 \leq card\ (C \cap (PR - kmin\ f\ PR))$
proof(*rule pigeonhole*)
show *finite PR* **by** *simp*
next
show $C \subseteq PR$ **by** *fact*
next
show $PR - kmin\ f\ PR \subseteq PR$ **by** *blast*

```

next
show  $\text{card } PR + (\text{khl} + 1) \leq \text{card } C + \text{card } (PR - \text{kmin } f PR)$ 
proof-
  from hmk and hCk have
     $np + \text{khl} < np + \text{card } C - \text{khl}$  by arith
  also
  from kmin-prop
  have ... =  $np + \text{card } C - \text{card } (\text{kmin } f PR)$ 
    by auto
  also
  have ... =  $\text{card } C + (\text{card } PR - \text{card } (\text{kmin } f PR))$ 
proof-
  from kmin-prop have
     $\text{card } (\text{kmin } f PR) \leq \text{card } PR$ 
    by (intro card-mono, auto)
  thus ?thesis by (simp)
qed
also
from kmin-prop
have ... =  $\text{card } C + \text{card } (PR - \text{kmin } f PR)$ 
proof-
  from kmin-prop and finite-kmin have
     $\text{card } PR - \text{card } (\text{kmin } f PR) = \text{card } (PR - \text{kmin } f PR)$ 
    by (intro card-Diff-subset[THEN sym])(auto)
  thus ?thesis by auto
qed
finally
show ?thesis
  by (simp)
qed
qed

have  $C \cap (PR - \text{kmin } f PR) \cap (PR - \text{kmax } g PR) \neq \{\}$ 
proof(intro subsets-int)
  show finite PR by simp
next
show  $C \cap (PR - \text{kmin } f PR) \subseteq PR$ 
  by blast
next
show  $PR - \text{kmax } g PR \subseteq PR$ 
  by blast
next
show  $\text{card } PR <$ 
   $\text{card } (C \cap (PR - \text{kmin } f PR)) + \text{card } (PR - \text{kmax } g PR)$ 
proof-
  from kmax-prop and finite-kmax
  have  $\text{card } (PR - \text{kmax } g PR) = \text{card } PR - \text{card } (\text{kmax } g PR)$ 
    by (intro card-Diff-subset, auto)
  with kmax-prop have

```

$\text{card } (PR - kmax\ g\ PR) = \text{card } PR - khl$ **by simp**
with b1
show ?thesis by arith
qed
qed

hence
 $\exists i. i \in C \wedge i \in (PR - kmin\ f\ PR) \wedge i \in (PR - kmax\ g\ PR)$
by blast
then obtain i where
in-C: $i \in C$ and
not-in-kmin: $i \in (PR - kmin\ f\ PR)$ and
not-in-kmax: $i \in (PR - kmax\ g\ PR)$ by blast
have $Min\ (\text{reduce } f\ PR) \leq f\ i$
proof(cases $i \in kmax\ f\ PR$)
case True
from reduce-not-empty and finite-reduce have
 $Min\ (\text{reduce } f\ PR) \in \text{reduce } f\ PR$ **by auto**
with True show ?thesis
using kmax-ge by blast

next
case False
with not-in-kmin
have $i \in PR - (kmax\ f\ PR \cup kmin\ f\ PR)$
by blast
with reduce-def have $f\ i \in \text{reduce } f\ PR$
by auto
with reduce-not-empty and finite-reduce
show ?thesis by auto
qed

moreover
have $g\ i \leq Max\ (\text{reduce } g\ PR)$
proof(cases $i \in kmin\ g\ PR$)
case True
from reduce-not-empty and finite-reduce have
 $Max\ (\text{reduce } g\ PR) \in \text{reduce } g\ PR$ **by auto**
with True show ?thesis
using kmin-le by blast

next
case False
with not-in-kmax
have $i \in PR - (kmax\ g\ PR \cup kmin\ g\ PR)$
by blast
with reduce-def have $g\ i \in \text{reduce } g\ PR$
by auto
with reduce-not-empty and finite-reduce
show ?thesis by auto
qed

moreover

note *in-C*
ultimately
show *?thesis* **by** *blast*
qed

2.3.2 Main theorem

The most part of this theorem can be proved with CVC-lite using the three previous lemmas (appendix A.2).

theorem *prec-enh*:

assumes

hC: $C \subseteq PR$ **and**
hCF: $np - nF \leq \text{card } C$ **and**
hFn: $3 * nF < np$ **and**
hFk: $nF = khl$ **and**
hbx: $\forall l \in C. |f l - g l| \leq x$ **and**
hby1: $\forall l \in C. \forall m \in C. |f l - f m| \leq y$ **and**
hby2: $\forall l \in C. \forall m \in C. |g l - g m| \leq y$ **and**
hpC: $p \in C$ **and**
hqC: $q \in C$

shows $|cfnl\ p\ f - cfnl\ q\ g| \leq y / 2 + x$

proof –

from *hCF* **and** *hFk*
have *hCk*: $np \leq \text{card } C + khl$ **by** *arith*
from *hFn* **and** *hFk*
have *hnk*: $3 * khl < np$ **by** *arith*
let *?maxf* = *Max* (*reduce f PR*)
and *?minf* = *Min* (*reduce f PR*)
and *?maxg* = *Max* (*reduce g PR*)
and *?ming* = *Min* (*reduce g PR*)
from *abs-distrib-div*
have $|cfnl\ p\ f - cfnl\ q\ g| =$
 $|?maxf + ?minf + - ?maxg + - ?ming| / 2$
by (*unfold cfnl-def, auto simp add: real-diff-def*)

moreover

have $|?maxf + ?minf + - ?maxg + - ?ming| \leq y + 2 * x$
 – The rest of the property can be proved by CVC-lite (see appendix A.2)

proof (*cases* $0 \leq ?maxf + ?minf + - ?maxg + - ?ming$)

case *True*

hence

$|?maxf + ?minf + - ?maxg + - ?ming| =$
 $?maxf + ?minf + - ?maxg + - ?ming$ **by** *arith*

moreover

from *uboundmax hC hCk*

obtain *mxf*

where *mxf*: $mxf \in C$ **and**

mxf: $?maxf \leq f\ mxf$ **by** *blast*

moreover

from *lboundmin hC hCk*

obtain mng
where $mnginC: mng \in C$ **and**
 $mng: g\ mng \leq ?ming$ **by** *blast*
moreover
from *same-bound hC hCk hnk*
obtain $m\ xn$
where $m\ xninC: m\ xn \in C$ **and**
 $m\ xnf: ?minf \leq f\ m\ xn$ **and**
 $m\ xng: g\ m\ xn \leq ?maxg$ **by** *blast*
ultimately
have
 $|?maxf + ?minf + -?maxg + -?ming| \leq$
 $(f\ m\ xf + -g\ mng) + (f\ m\ xn + -g\ m\ xn)$ **by** *arith*
also
from $m\ xninC\ hbx\ abs-le-D1$
have
 $\dots \leq (f\ m\ xf + -g\ mng) + x$
by *(auto simp add: real-diff-def)*
also
have
 $\dots = (f\ m\ xf + -f\ mng) + (f\ mng + -g\ mng) + x$
by *arith*
also
have $\dots \leq y + (f\ mng + -g\ mng) + x$
proof-
from $m\ xfinC\ mnginC\ hby1\ abs-le-D1$
have $f\ m\ xf + -f\ mng \leq y$
by *(auto simp add: real-diff-def)*
thus $?thesis$
by *(auto simp add: real-diff-def)*
qed
also
from $mnginC\ hbx\ abs-le-D1$
have $\dots \leq y + 2 * x$
by *(auto simp add: real-diff-def)*
finally
show $?thesis$.
next
case *False*
hence
 $|?maxf + ?minf + -?maxg + -?ming| =$
 $?maxg + ?ming + -?maxf + -?minf$ **by** *arith*
moreover
from *uboundmax hC hCk*
obtain $m\ xg$
where $m\ xginC: m\ xg \in C$ **and**
 $m\ xg: ?maxg \leq g\ m\ xg$ **by** *blast*
moreover
from *lboundmin hC hCk*

```

obtain mnf
  where mnfinC: mnf ∈ C and
    minf: f mnf ≤ ?minf by blast
moreover
from same-bound hC hCk hnk
obtain mzn
  where mzninC: mzn ∈ C and
    mznf: ?ming ≤ g mzn and
    mzng: f mzn ≤ ?maxf by blast
ultimately
have
  | ?maxf + ?minf + - ?maxg + - ?ming | ≤ =
  (g mxg + - f mnf) + (g mzn + - f mzn) by arith
also
from mzninC hbx
have ... ≤ = (g mxg + - f mnf) + x
  by (auto dest!: abs-le-D2)
also
have
  ... = (g mxg + - g mnf) + (g mnf + - f mnf) + x
  by arith
also
have ... ≤ = y + (g mnf + - f mnf) + x
proof-
  from mxginC mnfinC hby2 abs-le-D1
  have g mxg + - g mnf ≤ = y
    by (auto simp add: real-diff-def)
  thus ?thesis
    by (auto simp add: real-diff-def)
qed
also
from mnfinC hbx
have ... ≤ = y + 2 * x
  by (auto dest!: abs-le-D2)
finally
show ?thesis .
qed
ultimately
show ?thesis
  by simp
qed

```

2.4 Accuracy Preservation property

No new lemmas are needed for prove this property. The bound has been found using the lemmas *uboundmax* and *lboundmin*

This theorem can be proved with ICS and CVC-lite assuming those lemmas (see appendix A.3).

theorem *accur-pres*:

assumes

hC: $C \subseteq PR$ **and**

hCF: $np - nF \leq \text{card } C$ **and**

hFk: $nF = khl$ **and**

hby: $\forall l \in C. \forall m \in C. |f l - f m| \leq y$ **and**

hqC: $q \in C$

shows $|cfnl p f - f q| \leq y$

proof –

from *hCF* **and** *hFk*

have *npleCk*: $np \leq \text{card } C + khl$ **by** *arith*

show *?thesis*

proof(*cases f q ≤ cfnl p f*)

case *True*

from *npleCk hC* **and** *uboundmax*

have $\exists i \in C. \text{Max}(\text{reduce } f PR) \leq f i$

by *auto*

then obtain *pi* **where**

hpiC: $pi \in C$ **and**

fpiGeMax: $\text{Max}(\text{reduce } f PR) \leq f pi$ **by** *blast*

from *reduce-not-empty*

have $\text{Min}(\text{reduce } f PR) \leq \text{Max}(\text{reduce } f PR)$

by (*auto simp add: reduce-def*)

with *fpiGeMax* **have**

cfnlLeSpi: $cfnl p f \leq f pi$

by (*auto simp add: cfnl-def*)

with *True* **have**

$|cfnl p f - f q| \leq |f pi - f q|$

by *arith*

with *hpiC* **and** *hqC* **and** *hby* **show** *?thesis*

by *force*

next

case *False*

from *npleCk hC* **and** *lboundmin*

have $\exists i \in C. f i \leq \text{Min}(\text{reduce } f PR)$

by *auto*

then obtain *qi* **where**

hqiC: $qi \in C$ **and**

fqiLeMax: $f qi \leq \text{Min}(\text{reduce } f PR)$ **by** *blast*

from *reduce-not-empty*

have $\text{Min}(\text{reduce } f PR) \leq \text{Max}(\text{reduce } f PR)$

by (*auto simp add: reduce-def*)

with *fqiLeMax*

have $f qi \leq cfnl p f$

by (*auto simp add: cfnl-def*)

with *False* **have**

$|cfnl p f - f q| \leq |f qi - f q|$

by *arith*

with *hqiC* **and** *hqC* **and** *hby* **show** *?thesis*

```

    by force
  qed
qed
end

```

A CVC-lite and ICS proofs

A.1 Lemma `abs_distrib_div`

In the proof of the Fault-Tolerant Mid Point Algorithm we need to prove this simple lemma:

lemma *abs-distrib-div*:

$$0 < (c::real) \implies |a / c - b / c| = |a - b| / c$$

It is not possible to prove this lemma in Isabelle using *arith* nor *auto* tactics. Even if we added lemmas to the default simpset of HOL.

In the translation from Isabelle to ICS we need to change the division by a multiplication because this tools do not accept formulas with this arithmetic operator. Moreover, to translate the absolute value we define e constant for each application of that function. In ICS it is proved automatically.

File `abs_distrib_mult.ics`:

```

sig a: real.
sig b: real.
sig d: real.
prop abslhs := if [ a * d - b * d >= 0 ] then AL = a * d - b * d
               else AL = -(a * d - b * d) end.
prop absrhs := if [ a - b >= 0 ] then AR = a - b
               else AR = -(a - b) end.
prop asm := d >= 0.
prop concl := AL = AR * d.
sat ~[ [ abslhs & absrhs & asm ] => concl ].

```

It was not possible to find the proof in CVC-lite because the formula is not linear. Two proofs where attempted. In the first one we use lambda abstraction to define the absolute value. The second one is the same translation that we do in ICS.

File `abs_distrib_mult.cvc`:

```

% cvcl can't resolve because is a Non-linear arithmetic inequalities
a, b, c : REAL;
abs: REAL -> REAL = LAMBDA (x:REAL): IF x>=0 THEN x ELSE (-x) ENDIF;

```

```
QUERY (c >= 0 => abs(a*c-b*c) = abs(a-b)*c);
```

File `abs_distrib_mult2.cvc`:

```
% cvcl can't resolve because is a Non-linear arithmetic inequalities.
a, b, c, al, ar : REAL;
abslhs: BOOLEAN = IF (a * c - b * c >= 0) THEN (al = a * c - b * c)
ELSE (al = -(a * c - b * c)) ENDIF;
absrhs: BOOLEAN = IF (a - b >= 0) THEN (ar = a - b)
ELSE (ar = -(a - b)) ENDIF;

ASSERT(c >= 0);
QUERY ((abslhs AND absrhs) => (al = ar * c));
```

A.2 Bound for Precision Enhancement property

In order to prove Precision Enhancement for Lynch's algorithm we need to prove that:

$$\text{have } |Max(\text{reduce } f \text{ PR}) + Min(\text{reduce } f \text{ PR}) + \\ - Max(\text{reduce } g \text{ PR}) + - Min(\text{reduce } g \text{ PR})| \leq y + 2 * x$$

This is the result of the whole theorem where we multiply by two both sides of the inequality.

In order to do the proof we need to translate also the lemmas *uboundmax*, *lboundmin*, *same_bound* (lemmas about the existence of some bounds), the axiom *constants_ax* and the assumptions of the theorem.

We make five different translations. In each one we where increasing the amount of eliminated quantifiers.

File `bound_prec_enh4.cvc`:

```
% Version removing de universal quantification over C
% and skolemising (creating new variables)
% and removing universal quantifiers in hbx
% It work :)

SETPROC : TYPE;
PROC : TYPE;

pmaxf, pmaxg, pminf, pming : PROC;
sbgf, sbgf: PROC;

np, khl: INT;
maxreduc: (PROC -> REAL, SETPROC) -> REAL;
minreduc: (PROC -> REAL, SETPROC) -> REAL;
```

```

x, y : REAL;

f : PROC -> REAL;
g : PROC -> REAL;

PR, C : SETPROC;

card : SETPROC -> INT;
INCL : (SETPROC,SETPROC) -> BOOLEAN;
INSET : (PROC, SETPROC) -> BOOLEAN;

abs: REAL -> REAL = LAMBDA (x:REAL): IF x>=0 THEN x ELSE (-x) ENDIF;

constants_ax: BOOLEAN = 2*khl < np AND khl >= 0;
hcard: BOOLEAN = card(C) >= 0;

uboundmaxf: BOOLEAN =
    INCL(C,PR) AND np <= card(C) + khl
    => INSET(pmaxf,C) AND maxreduc( f, PR) <= f(pmaxf);

uboundmaxg: BOOLEAN =
    INCL(C,PR) AND np <= card(C) + khl
    => INSET(pmaxg,C) AND maxreduc( g, PR) <= g(pmaxg);

lboundminf: BOOLEAN =
    INCL(C,PR) AND np <= card(C) + khl
    => INSET(pminf,C) AND minreduc( f, PR) >= f(pminf);

lboundming: BOOLEAN =
    INCL(C,PR) AND np <= card(C) + khl
    => INSET(pming,C) AND minreduc( g, PR) >= g(pming);

same_bound_f_g: BOOLEAN =
    INCL(C,PR) AND np <= card(C) + khl AND 3*khl < np
    => INSET(sbfg,C) AND minreduc( f, PR) <= f(sbfg)
    AND maxreduc( g, PR) >= g(sbfg);

same_bound_g_f: BOOLEAN =
    INCL(C,PR) AND np <= card(C) + khl AND 3*khl < np
    => INSET(sbgf,C) AND minreduc( g, PR) <= g(sbgf)
    AND maxreduc( f, PR) >= f(sbgf);

hC : BOOLEAN = INCL(C,PR);

```

```

hnp : BOOLEAN = np <= card(C) + khl AND 3*khl < np;

% hbx : BOOLEAN = FORALL (l:PROC): INSET(l,C) => abs(f(l) - g(l)) <= x;

hbx_pmaxf : BOOLEAN = INSET(pmaxf,C) => abs(f(pmaxf) - g(pmaxf)) <= x;
hbx_pmaxg : BOOLEAN = INSET(pmaxg,C) => abs(f(pmaxg) - g(pmaxg)) <= x;
hbx_pminf : BOOLEAN = INSET(pminf,C) => abs(f(pminf) - g(pminf)) <= x;
hbx_pming : BOOLEAN = INSET(pming,C) => abs(f(pming) - g(pming)) <= x;
hbx_sbf g : BOOLEAN = INSET(sbf g,C) => abs(f(sbf g) - g(sbf g)) <= x;
hbx_sbf g : BOOLEAN = INSET(sbf g,C) => abs(f(sbf g) - g(sbf g)) <= x;

hby1 : BOOLEAN = FORALL (l:PROC): INSET(l,C) =>
    FORALL (m:PROC): INSET(m,C) => abs(f(l) - f(m)) <= y;

hby2 : BOOLEAN = FORALL (l:PROC): INSET(l,C) =>
    FORALL (m:PROC): INSET(m,C) => abs(g(l) - g(m)) <= y;

ASSERT(hcard AND constants_ax AND hC AND hnp AND
uboundmaxf AND uboundmaxg AND
lboundminf AND lboundming AND
same_bound_f_g AND same_bound_g_f AND
    hbx_pmaxf AND hbx_pmaxg AND hbx_pminf AND
    hbx_pming AND hbx_sbf g AND hbx_sbf g AND
    hby1 AND hby2);

QUERY( abs(maxreduc(f,PR) + minreduc(f,PR)
    - maxreduc(g,PR) - minreduc(g,PR)) <=
    y + 2 * x);

% DUMP_PROOF;

Note that we leave quantifiers in some assumptions.
In the next file we also try to do the proof with all quantifiers, but CVC
cannot find it.
File bound_prec_enh.cvc:

% Version with quantifiers
% do not work :(
% Finish saying:
% Unknown.
% CVC Lite was incomplete in this example due to:
% * Non-linear arithmetic inequalities

```

```

SETPROC : TYPE;
PROC : TYPE;

np, khl: INT;
maxreduc: (PROC -> REAL, SETPROC) -> REAL;
minreduc: (PROC -> REAL, SETPROC) -> REAL;

x, y : REAL;

f : PROC -> REAL;
g : PROC -> REAL;

PR, C : SETPROC;

card : SETPROC -> INT;
INCL : (SETPROC,SETPROC) -> BOOLEAN;
INSET : (PROC, SETPROC) -> BOOLEAN;

abs: REAL -> REAL = LAMBDA (x:REAL): IF x>=0 THEN x ELSE (-x) ENDIF;

constants_ax: BOOLEAN = 2*khl < np AND khl >= 0;
hcard: BOOLEAN = card(C) >= 0;

uboundmaxf: BOOLEAN = FORALL (C : SETPROC):
    INCL(C,PR) AND np <= card(C) + khl
    => EXISTS (i:PROC): INSET(i,C) AND maxreduc( f, PR) <= f(i);

uboundmaxg: BOOLEAN = FORALL (C : SETPROC):
    INCL(C,PR) AND np <= card(C) + khl
    => EXISTS (i:PROC): INSET(i,C) AND maxreduc( g, PR) <= g(i);

lboundminf: BOOLEAN = FORALL (C : SETPROC):
    INCL(C,PR) AND np <= card(C) + khl
    => EXISTS (i:PROC): INSET(i,C) AND minreduc( f, PR) >= f(i);

lboundming: BOOLEAN = FORALL (C : SETPROC):
    INCL(C,PR) AND np <= card(C) + khl
    => EXISTS (i:PROC): INSET(i,C) AND minreduc( g, PR) >= g(i);

same_bound_f_g: BOOLEAN = FORALL (C : SETPROC):
    INCL(C,PR) AND np <= card(C) + khl AND 3*khl < np
    => EXISTS (i:PROC): INSET(i,C) AND minreduc( f, PR) <= f(i)
    AND maxreduc( g, PR) >= g(i);

```

```

same_bound_g_f: BOOLEAN = FORALL (C : SETPROC):
    INCL(C,PR) AND np <= card(C) + khl AND 3*khl < np
    => EXISTS (i:PROC): INSET(i,C) AND minreduc( g, PR) <= g(i)
    AND maxreduc( f, PR) >= f(i);

hC : BOOLEAN = INCL(C,PR);

hnp : BOOLEAN = np <= card(C) + khl AND 3*khl < np;

hbx : BOOLEAN = FORALL (l:PROC): INSET(l,C) => abs(f(l) - g(l)) <= x;

hby1 : BOOLEAN = FORALL (l:PROC): INSET(l,C) =>
    FORALL (m:PROC): INSET(m,C) => abs(f(l) - f(m)) <= y;

hby2 : BOOLEAN = FORALL (l:PROC): INSET(l,C) =>
    FORALL (m:PROC): INSET(m,C) => abs(g(l) - g(m)) <= y;

ASSERT(hcard AND constants_ax AND hC AND hnp AND
uboundmaxf AND uboundmaxg AND lboundminf AND lboundming
AND same_bound_f_g AND same_bound_g_f
    AND hC AND hbx AND hby1 AND hby2);

QUERY( abs(maxreduc(f,PR) + minreduc(f,PR)
    - maxreduc(g,PR) - minreduc(g,PR)) <=
    y + 2 * x);

```

We also try to do the proof removing all quantifiers and the proof was successful.

File bound_prec_enh7.cvc:

```

% Version removing de universal quantification over C
% and skolemising (creating new variables)
% and removing universal quantifiers in hbx and hby
% It work :)

SETPROC : TYPE;
PROC : TYPE;

np, khl: INT;
maxreduc: (PROC -> REAL, SETPROC) -> REAL;
minreduc: (PROC -> REAL, SETPROC) -> REAL;

```

```

x, y : REAL;

f : PROC -> REAL;
g : PROC -> REAL;
pmaxf, pmaxg, pminf, pming : PROC;
sbf, sbgf : PROC;

PR, C : SETPROC;

card : SETPROC -> INT;
INCL : (SETPROC,SETPROC) -> BOOLEAN;
INSET : (PROC, SETPROC) -> BOOLEAN;

abs: REAL -> REAL = LAMBDA (x:REAL): IF (x>=0) THEN x ELSE (-x) ENDIF;

constants_ax: BOOLEAN = 2*khl < np AND khl >= 0;
hC : BOOLEAN = INCL(C,PR);
hnp : BOOLEAN = np <= card(C) + khl AND 3*khl < np;
hcard: BOOLEAN = card(C) >= 0; % redundant

uboundmaxf: BOOLEAN =
    ( maxreduc( f, PR) <= f(pmaxf));

uboundmaxg: BOOLEAN =
    (maxreduc( g, PR) <= g(pmaxg));

lboundminf: BOOLEAN =
    (minreduc( f, PR) >= f(pminf));

lboundming: BOOLEAN =
    ( minreduc( g, PR) >= g(pming));

same_bound_f_g: BOOLEAN =
    ( minreduc( f, PR) <= f(sbf)
    AND maxreduc( g, PR) >= g(sbf));

same_bound_g_f: BOOLEAN =
    ( minreduc( g, PR) <= g(sbgf)
    AND maxreduc( f, PR) >= f(sbgf));

%hbx : BOOLEAN = FORALL (l:PROC): INSET(l,C) => abs(f(l) - g(l)) <= x;

hbx_pmaxf : BOOLEAN = abs(f(pmaxf) - g(pmaxf)) <= x;

```

```

hbx_pmaxg : BOOLEAN = abs(f(pmaxg) - g(pmaxg)) <= x;
hbx_pminf : BOOLEAN = abs(f(pminf) - g(pminf)) <= x;
hbx_pming : BOOLEAN = abs(f(pming) - g(pming)) <= x;
hbx_sbfq : BOOLEAN = abs(f(sbfq) - g(sbfq)) <= x;
hbx_sbgf : BOOLEAN = abs(f(sbgf) - g(sbgf)) <= x;

%hby1 : BOOLEAN = FORALL (l:PROC): INSET(l,C) =>
%
%           FORALL (m:PROC): INSET(m,C) => abs(f(l) - f(m)) <= y;

hby1_pmaxf_pmaxg : BOOLEAN = (abs(f(pmaxf) - f(pmaxg)) <= y);
hby1_pmaxf_pminf : BOOLEAN = (abs(f(pmaxf) - f(pminf)) <= y);
hby1_pmaxf_pming : BOOLEAN = (abs(f(pmaxf) - f(pming)) <= y);
hby1_pmaxf_sbfq : BOOLEAN = (abs(f(pmaxf) - f(sbfq)) <= y);
hby1_pmaxf_sbgf : BOOLEAN = (abs(f(pmaxf) - f(sbgf)) <= y);

hby1_pmaxg_pminf : BOOLEAN = (abs(f(pmaxg) - f(pminf)) <= y);
hby1_pmaxg_pming : BOOLEAN = (abs(f(pmaxg) - f(pming)) <= y);
hby1_pmaxg_sbfq : BOOLEAN = (abs(f(pmaxg) - f(sbfq)) <= y);
hby1_pmaxg_sbgf : BOOLEAN = (abs(f(pmaxg) - f(sbgf)) <= y);

hby1_pminf_pming : BOOLEAN = (abs(f(pminf) - f(pming)) <= y);
hby1_pminf_sbfq : BOOLEAN = (abs(f(pminf) - f(sbfq)) <= y);
hby1_pminf_sbgf : BOOLEAN = (abs(f(pminf) - f(sbgf)) <= y);

hby1_pming_sbfq : BOOLEAN = (abs(f(pming) - f(sbfq)) <= y);
hby1_pming_sbgf : BOOLEAN = (abs(f(pming) - f(sbgf)) <= y);

hby1_sbfq_sbgf : BOOLEAN = (abs(f(sbfq) - f(sbgf)) <= y);

hby2_pmaxf_pmaxg : BOOLEAN = (abs(g(pmaxf) - g(pmaxg)) <= y);
hby2_pmaxf_pminf : BOOLEAN = (abs(g(pmaxf) - g(pminf)) <= y);
hby2_pmaxf_pming : BOOLEAN = (abs(g(pmaxf) - g(pming)) <= y);
hby2_pmaxf_sbfq : BOOLEAN = (abs(g(pmaxf) - g(sbfq)) <= y);
hby2_pmaxf_sbgf : BOOLEAN = (abs(g(pmaxf) - g(sbgf)) <= y);

hby2_pmaxg_pminf : BOOLEAN = (abs(g(pmaxg) - g(pminf)) <= y);
hby2_pmaxg_pming : BOOLEAN = (abs(g(pmaxg) - g(pming)) <= y);
hby2_pmaxg_sbfq : BOOLEAN = (abs(g(pmaxg) - g(sbfq)) <= y);
hby2_pmaxg_sbgf : BOOLEAN = (abs(g(pmaxg) - g(sbgf)) <= y);

hby2_pminf_pming : BOOLEAN = (abs(g(pminf) - g(pming)) <= y);
hby2_pminf_sbfq : BOOLEAN = (abs(g(pminf) - g(sbfq)) <= y);

```

```

hby2_pminf_sbgf : BOOLEAN = (abs(g(pminf) - g(sbgf)) <= y);

hby2_pming_sbgf : BOOLEAN = (abs(g(pming) - g(sbgf)) <= y);
hby2_pming_sbgf : BOOLEAN = (abs(g(pming) - g(sbgf)) <= y);

hby2_sbgf_sbgf : BOOLEAN = (abs(g(sbgf) - g(sbgf)) <= y);

WHERE;
PUSH;
ASSERT( hcard AND constants_ax AND hC AND hnp AND
uboundmaxf AND uboundmaxg AND lboundminf AND lboundming AND
same_bound_f_g AND same_bound_g_f AND
        hbx_pmaxf AND hbx_pmaxg AND hbx_pminf AND
        hbx_pming AND hbx_sbgf AND hbx_sbgf AND
hby1_pmaxf_pmaxg AND hby1_pmaxf_pminf AND hby1_pmaxf_pming AND
hby1_pmaxf_sbgf AND hby1_pmaxf_sbgf AND
hby1_pmaxg_pminf AND hby1_pmaxg_pming AND
        hby1_pmaxg_sbgf AND hby1_pmaxg_sbgf AND
hby1_pminf_pming AND hby1_pminf_sbgf AND hby1_pminf_sbgf AND
        hby1_pming_sbgf AND hby1_pming_sbgf AND hby1_sbgf_sbgf AND
hby2_pmaxf_pmaxg AND hby2_pmaxf_pminf AND hby2_pmaxf_pming AND
hby2_pmaxf_sbgf AND hby2_pmaxf_sbgf AND
hby2_pmaxg_pminf AND hby2_pmaxg_pming AND
        hby2_pmaxg_sbgf AND hby2_pmaxg_sbgf AND
hby2_pminf_pming AND hby2_pminf_sbgf AND hby2_pminf_sbgf AND
        hby2_pming_sbgf AND hby2_pming_sbgf AND hby2_sbgf_sbgf);

QUERY( abs(maxreduc(f,PR) + minreduc(f,PR)
        - maxreduc(g,PR) - minreduc(g,PR)) <=
        y + 2 * x);
POP;

PUSH;
WHERE;

ASSERT( uboundmaxf AND uboundmaxg AND lboundminf AND lboundming AND
same_bound_f_g AND same_bound_g_f AND
        hbx_pmaxf AND hbx_pmaxg AND hbx_pminf AND
        hbx_pming AND hbx_sbgf AND hbx_sbgf AND
hby1_pmaxf_pmaxg AND hby1_pmaxf_pminf AND hby1_pmaxf_pming AND
hby1_pmaxf_sbgf AND hby1_pmaxf_sbgf AND
hby1_pmaxg_pminf AND hby1_pmaxg_pming AND
        hby1_pmaxg_sbgf AND hby1_pmaxg_sbgf AND
hby1_pminf_pming AND hby1_pminf_sbgf AND hby1_pminf_sbgf AND

```

```

        hby1_pming_sbf AND hby1_pming_sbf AND hby1_sbf_sbf AND
hby2_pmaxf_pmaxg AND hby2_pmaxf_pminf AND hby2_pmaxf_pming);

```

```

QUERY( abs(maxreduc(f,PR) + minreduc(f,PR)
        - maxreduc(g,PR) - minreduc(g,PR)) <=
        y + 2 * x);

```

```
POP;
```

```

PUSH;
WHERE;

```

```
% this do not work
```

```

ASSERT( uboundmaxg AND lboundminf AND lboundming AND
same_bound_f_g AND same_bound_g_f AND
        hbx_pmaxf AND hbx_pmaxg AND hbx_pminf AND
        hbx_pming AND hbx_sbf AND hbx_sbf AND
hby1_pmaxf_pmaxg AND hby1_pmaxf_pminf AND hby1_pmaxf_pming AND
hby1_pmaxf_sbf AND hby1_pmaxf_sbf AND
hby1_pmaxg_pminf AND hby1_pmaxg_pming AND
        hby1_pmaxg_sbf AND hby1_pmaxg_sbf AND
hby1_pminf_pming AND hby1_pminf_sbf AND hby1_pminf_sbf AND
        hby1_pming_sbf AND hby1_pming_sbf AND hby1_sbf_sbf AND
hby2_pmaxf_pmaxg AND hby2_pmaxf_pminf AND hby2_pmaxf_pming);

```

```

QUERY( abs(maxreduc(f,PR) + minreduc(f,PR)
        - maxreduc(g,PR) - minreduc(g,PR)) <=
        y + 2 * x);

```

```
POP;
```

```

%DUMP_PROOF;
%WHERE;
%COUNTEREXAMPLE;

```

From this last file we make the translation also for ICS adding a constant for each application of the absolute value. In this case ICS do not find the proof.

File bound_prec_enh.ics:

```

% Translation from bound_prec_enh6.cvc
% It not stop :(

```

```

sig np: int.
sig khl: int.

```

```

sig maxreducf: real.
sig minreducf: real.
sig maxreducg: real.
sig minreducg: real.

sig x: real.
sig y: real.

sig f_pmaxf: real.
sig f_pmaxg: real.
sig f_pminf: real.
sig f_pming: real.
sig f_sbf: real.
sig f_sbgf: real.

sig g_pmaxf: real.
sig g_pmaxg: real.
sig g_pminf: real.
sig g_pming: real.
sig g_sbf: real.
sig g_sbgf: real.

% f : PROC -> REAL;
% g : PROC -> REAL;
% pmaxf, pmaxg, pminf, pming : PROC;
% sbf, sbgf: PROC;

% PR, C : SETPROC;

sig card_C : int.

% INCL : (SETPROC,SETPROC) -> BOOLEAN;
% INSET : (PROC, SETPROC) -> BOOLEAN;

prop constants_ax := 2*khl < np & khl >= 0.
prop hC := INCL_C_PR.
prop hnp := np <= card_C + khl & 3*khl < np.
prop hcard := card_C >= 0.

prop uboundmaxf :=
  [INCL_C_PR & np <= card_C + khl]

```

```

=> [INSET_pmaxf_C & maxreducf <= f_pmaxf].

prop uboundmaxg :=
  [INCL_C_PR & np <= card_C + khl]
  => [INSET_pmaxg_C & maxreducg <= g_pmaxg].

prop lboundminf :=
  [INCL_C_PR & np <= card_C + khl]
  => [INSET_pminf_C & minreducf >= f_pminf].

prop lboundming :=
  [INCL_C_PR & np <= card_C + khl]
  => [INSET_pming_C & minreducg >= g_pming].

prop same_bound_f_g :=
  [INCL_C_PR & np <= card_C + khl & 3*khl < np]
  => [INSET_sbf_C & minreducf <= f_sbf_C
  & maxreducg >= g_sbf_C].

prop same_bound_g_f :=
  [INCL_C_PR & np <= card_C + khl & 3*khl < np]
  => [INSET_sbgf_C & minreducg <= g_sbgf_C
  & maxreducf >= f_sbgf_C].

prop hbx_pmaxf := INSET_pmaxf_C => abs_f_pmaxf_g_pmaxf <= x.
prop hbx_pmaxg := INSET_pmaxg_C => abs_f_pmaxg_g_pmaxg <= x.
prop hbx_pminf := INSET_pminf_C => abs_f_pminf_g_pminf <= x.
prop hbx_pming := INSET_pming_C => abs_f_pming_g_pming <= x.
prop hbx_sbf_C := INSET_sbf_C => abs_f_sbf_g_sbf <= x.
prop hbx_sbgf_C := INSET_sbgf_C => abs_f_sbgf_g_sbgf <= x.

prop hby1_pmaxf_pmaxg := INSET_pmaxf_C =>
  [INSET_pmaxg_C => abs_f_pmaxf_f_pmaxg <= y].
prop hby1_pmaxf_pminf := INSET_pmaxf_C =>
  [INSET_pminf_C => abs_f_pmaxf_f_pminf <= y].
prop hby1_pmaxf_pming := INSET_pmaxf_C =>
  [INSET_pming_C => abs_f_pmaxf_f_pming <= y].
prop hby1_pmaxf_sbf_C := INSET_pmaxf_C =>
  [INSET_sbf_C => abs_f_pmaxf_f_sbf <= y].
prop hby1_pmaxf_sbgf_C := INSET_pmaxf_C =>
  [INSET_sbgf_C => abs_f_pmaxf_f_sbgf <= y].

prop hby1_pmaxg_pminf := INSET_pmaxg_C =>
  [INSET_pminf_C => abs_f_pmaxg_f_pminf <= y].

```

```

prop hby1_pmaxg_pming := INSET_pmaxg_C =>
    [INSET_pming_C => abs_f_pmaxg_f_pming <= y].
prop hby1_pmaxg_sbfq := INSET_pmaxg_C =>
    [INSET_sbfq_C => abs_f_pmaxg_f_sbfq <= y].
prop hby1_pmaxg_sbgf := INSET_pmaxg_C =>
    [INSET_sbgf_C => abs_f_pmaxg_f_sbgf <= y].

prop hby1_pminf_pming := INSET_pminf_C =>
    [INSET_pming_C => abs_f_pminf_f_pming <= y].
prop hby1_pminf_sbfq := INSET_pminf_C =>
    [INSET_sbfq_C => abs_f_pminf_f_sbfq <= y].
prop hby1_pminf_sbgf := INSET_pminf_C =>
    [INSET_sbgf_C => abs_f_pminf_f_sbgf <= y].

prop hby1_pming_sbfq := INSET_pming_C =>
    [INSET_sbfq_C => abs_f_pming_f_sbfq <= y].
prop hby1_pming_sbgf := INSET_pming_C =>
    [INSET_sbgf_C => abs_f_pming_f_sbgf <= y].

prop hby1_sbfq_sbgf := INSET_sbfq_C =>
    [INSET_sbgf_C => abs_f_sbfq_f_sbgf <= y].

prop hby2_pmaxf_pmaxg := INSET_pmaxf_C =>
    [INSET_pmaxg_C => abs_g_pmaxf_g_pmaxg <= y].
prop hby2_pmaxf_pminf := INSET_pmaxf_C =>
    [INSET_pminf_C => abs_g_pmaxf_g_pminf <= y].
prop hby2_pmaxf_pming := INSET_pmaxf_C =>
    [INSET_pming_C => abs_g_pmaxf_g_pming <= y].
prop hby2_pmaxf_sbfq := INSET_pmaxf_C =>
    [INSET_sbfq_C => abs_g_pmaxf_g_sbfq <= y].
prop hby2_pmaxf_sbgf := INSET_pmaxf_C =>
    [INSET_sbgf_C => abs_g_pmaxf_g_sbgf <= y].

prop hby2_pmaxg_pminf := INSET_pmaxg_C =>
    [INSET_pminf_C => abs_g_pmaxg_g_pminf <= y].
prop hby2_pmaxg_pming := INSET_pmaxg_C =>
    [INSET_pming_C => abs_g_pmaxg_g_pming <= y].
prop hby2_pmaxg_sbfq := INSET_pmaxg_C =>
    [INSET_sbfq_C => abs_g_pmaxg_g_sbfq <= y].
prop hby2_pmaxg_sbgf := INSET_pmaxg_C =>
    [INSET_sbgf_C => abs_g_pmaxg_g_sbgf <= y].

prop hby2_pminf_pming := INSET_pminf_C =>

```

```

[INSET_pming_C => abs_g_pminf_g_pming <= y].
prop hby2_pminf_sbfq := INSET_pminf_C =>
[INSET_sbfq_C => abs_g_pminf_g_sbfq <= y].
prop hby2_pminf_sbgf := INSET_pminf_C =>
[INSET_sbgf_C => abs_g_pminf_g_sbgf <= y].

prop hby2_pming_sbfq := INSET_pming_C =>
[INSET_sbfq_C => abs_g_pming_g_sbfq <= y].
prop hby2_pming_sbgf := INSET_pming_C =>
[INSET_sbgf_C => abs_g_pming_g_sbgf <= y].

prop hby2_sbfq_sbgf := INSET_sbfq_C =>
[INSET_sbgf_C => abs_g_sbfq_g_sbgf <= y].

% abs: REAL -> REAL = LAMBDA (x:REAL): IF x>=0 THEN x ELSE (-x) ENDIF;

prop p_abs_f_pmaxf_g_pmaxf := if [ f_pmaxf - g_pmaxf >= 0 ]
then abs_f_pmaxf_g_pmaxf = f_pmaxf - g_pmaxf
else abs_f_pmaxf_g_pmaxf = -(f_pmaxf - g_pmaxf)
end.
prop p_abs_f_pmaxg_g_pmaxg := if [ f_pmaxg - g_pmaxg >= 0 ]
then abs_f_pmaxg_g_pmaxg = f_pmaxg - g_pmaxg
else abs_f_pmaxg_g_pmaxg = -(f_pmaxg - g_pmaxg)
end.
prop p_abs_f_pminf_g_pminf := if [ f_pminf - g_pminf >= 0 ]
then abs_f_pminf_g_pminf = f_pminf - g_pminf
else abs_f_pminf_g_pminf = -(f_pminf - g_pminf)
end.
prop p_abs_f_pming_g_pming := if [ f_pming - g_pming >= 0 ]
then abs_f_pming_g_pming = f_pming - g_pming
else abs_f_pming_g_pming = -(f_pming - g_pming)
end.
prop p_abs_f_sbfq_g_sbfq := if [ f_sbfq - g_sbfq >= 0 ]
then abs_f_sbfq_g_sbfq = f_sbfq - g_sbfq
else abs_f_sbfq_g_sbfq = -(f_sbfq - g_sbfq)
end.
prop p_abs_f_sbgf_g_sbgf := if [ f_sbgf - g_sbgf >= 0 ]
then abs_f_sbgf_g_sbgf = f_sbgf - g_sbgf
else abs_f_sbgf_g_sbgf = -(f_sbgf - g_sbgf)
end.

prop p_abs_f_pmaxf_f_pmaxg := if [ f_pmaxf - f_pmaxg >= 0 ]

```

```

        then abs_f_pmaxf_f_pmaxg = f_pmaxf - f_pmaxg
        else abs_f_pmaxf_f_pmaxg = -(f_pmaxf - f_pmaxg)
        end.
prop p_abs_f_pmaxf_f_pminf := if [ f_pmaxf - f_pminf >= 0 ]
    then abs_f_pmaxf_f_pminf = f_pmaxf - f_pminf
    else abs_f_pmaxf_f_pminf = -(f_pmaxf - f_pminf)
    end.
prop p_abs_f_pmaxf_f_pming := if [ f_pmaxf - f_pming >= 0 ]
    then abs_f_pmaxf_f_pming = f_pmaxf - f_pming
    else abs_f_pmaxf_f_pming = -(f_pmaxf - f_pming)
    end.
prop p_abs_f_pmaxf_f_sbfg := if [ f_pmaxf - f_sbfg >= 0 ]
    then abs_f_pmaxf_f_sbfg = f_pmaxf - f_sbfg
    else abs_f_pmaxf_f_sbfg = -(f_pmaxf - f_sbfg)
    end.
prop p_abs_f_pmaxf_f_sbgf := if [ f_pmaxf - f_sbgf >= 0 ]
    then abs_f_pmaxf_f_sbgf = f_pmaxf - f_sbgf
    else abs_f_pmaxf_f_sbgf = -(f_pmaxf - f_sbgf)
    end.

prop p_abs_f_pmaxg_f_pminf := if [ f_pmaxg - f_pminf >= 0 ]
    then abs_f_pmaxg_f_pminf = f_pmaxg - f_pminf
    else abs_f_pmaxg_f_pminf = -(f_pmaxg - f_pminf)
    end.
prop p_abs_f_pmaxg_f_pming := if [ f_pmaxg - f_pming >= 0 ]
    then abs_f_pmaxg_f_pming = f_pmaxg - f_pming
    else abs_f_pmaxg_f_pming = -(f_pmaxg - f_pming)
    end.
prop p_abs_f_pmaxg_f_sbfg := if [ f_pmaxg - f_sbfg >= 0 ]
    then abs_f_pmaxg_f_sbfg = f_pmaxg - f_sbfg
    else abs_f_pmaxg_f_sbfg = -(f_pmaxg - f_sbfg)
    end.
prop p_abs_f_pmaxg_f_sbgf := if [ f_pmaxg - f_sbgf >= 0 ]
    then abs_f_pmaxg_f_sbgf = f_pmaxg - f_sbgf
    else abs_f_pmaxg_f_sbgf = -(f_pmaxg - f_sbgf)
    end.

prop p_abs_f_pminf_f_pming := if [ f_pminf - f_pming >= 0 ]
    then abs_f_pminf_f_pming = f_pminf - f_pming
    else abs_f_pminf_f_pming = -(f_pminf - f_pming)
    end.
prop p_abs_f_pminf_f_sbfg := if [ f_pminf - f_sbfg >= 0 ]
    then abs_f_pminf_f_sbfg = f_pminf - f_sbfg
    else abs_f_pminf_f_sbfg = -(f_pminf - f_sbfg)
    end.

```

```

end.
prop p_abs_f_pminf_f_sbgf := if [ f_pminf - f_sbgf >= 0 ]
  then abs_f_pminf_f_sbgf = f_pminf - f_sbgf
  else abs_f_pminf_f_sbgf = -(f_pminf - f_sbgf)
end.

prop p_abs_f_pming_f_sbgf := if [ f_pming - f_sbgf >= 0 ]
  then abs_f_pming_f_sbgf = f_pming - f_sbgf
  else abs_f_pming_f_sbgf = -(f_pming - f_sbgf)
end.
prop p_abs_f_pming_f_sbgf := if [ f_pming - f_sbgf >= 0 ]
  then abs_f_pming_f_sbgf = f_pming - f_sbgf
  else abs_f_pming_f_sbgf = -(f_pming - f_sbgf)
end.

prop p_abs_f_sbgf_f_sbgf := if [ f_sbgf - f_sbgf >= 0 ]
  then abs_f_sbgf_f_sbgf = f_sbgf - f_sbgf
  else abs_f_sbgf_f_sbgf = -(f_sbgf - f_sbgf)
end.

% ***** %

prop p_abs_g_pmaxf_g_pmaxg := if [ g_pmaxf - g_pmaxg >= 0 ]
  then abs_g_pmaxf_g_pmaxg = g_pmaxf - g_pmaxg
  else abs_g_pmaxf_g_pmaxg = -(g_pmaxf - g_pmaxg)
end.
prop p_abs_g_pmaxf_g_pminf := if [ g_pmaxf - g_pminf >= 0 ]
  then abs_g_pmaxf_g_pminf = g_pmaxf - g_pminf
  else abs_g_pmaxf_g_pminf = -(g_pmaxf - g_pminf)
end.
prop p_abs_g_pmaxf_g_pming := if [ g_pmaxf - g_pming >= 0 ]
  then abs_g_pmaxf_g_pming = g_pmaxf - g_pming
  else abs_g_pmaxf_g_pming = -(g_pmaxf - g_pming)
end.
prop p_abs_g_pmaxf_g_sbgf := if [ g_pmaxf - g_sbgf >= 0 ]
  then abs_g_pmaxf_g_sbgf = g_pmaxf - g_sbgf
  else abs_g_pmaxf_g_sbgf = -(g_pmaxf - g_sbgf)
end.
prop p_abs_g_pmaxf_g_sbgf := if [ g_pmaxf - g_sbgf >= 0 ]
  then abs_g_pmaxf_g_sbgf = g_pmaxf - g_sbgf
  else abs_g_pmaxf_g_sbgf = -(g_pmaxf - g_sbgf)
end.

prop p_abs_g_pmaxg_g_pminf := if [ g_pmaxg - g_pminf >= 0 ]

```

```

        then abs_g_pmaxg_g_pminf = g_pmaxg - g_pminf
        else abs_g_pmaxg_g_pminf = -(g_pmaxg - g_pminf)
        end.
prop p_abs_g_pmaxg_g_pming := if [ g_pmaxg - g_pming >= 0 ]
    then abs_g_pmaxg_g_pming = g_pmaxg - g_pming
    else abs_g_pmaxg_g_pming = -(g_pmaxg - g_pming)
    end.
prop p_abs_g_pmaxg_g_sbfm := if [ g_pmaxg - g_sbfm >= 0 ]
    then abs_g_pmaxg_g_sbfm = g_pmaxg - g_sbfm
    else abs_g_pmaxg_g_sbfm = -(g_pmaxg - g_sbfm)
    end.
prop p_abs_g_pmaxg_g_sbgf := if [ g_pmaxg - g_sbgf >= 0 ]
    then abs_g_pmaxg_g_sbgf = g_pmaxg - g_sbgf
    else abs_g_pmaxg_g_sbgf = -(g_pmaxg - g_sbgf)
    end.

prop p_abs_g_pminf_g_pming := if [ g_pminf - g_pming >= 0 ]
    then abs_g_pminf_g_pming = g_pminf - g_pming
    else abs_g_pminf_g_pming = -(g_pminf - g_pming)
    end.
prop p_abs_g_pminf_g_sbfm := if [ g_pminf - g_sbfm >= 0 ]
    then abs_g_pminf_g_sbfm = g_pminf - g_sbfm
    else abs_g_pminf_g_sbfm = -(g_pminf - g_sbfm)
    end.
prop p_abs_g_pminf_g_sbgf := if [ g_pminf - g_sbgf >= 0 ]
    then abs_g_pminf_g_sbgf = g_pminf - g_sbgf
    else abs_g_pminf_g_sbgf = -(g_pminf - g_sbgf)
    end.

prop p_abs_g_pming_g_sbfm := if [ g_pming - g_sbfm >= 0 ]
    then abs_g_pming_g_sbfm = g_pming - g_sbfm
    else abs_g_pming_g_sbfm = -(g_pming - g_sbfm)
    end.
prop p_abs_g_pming_g_sbgf := if [ g_pming - g_sbgf >= 0 ]
    then abs_g_pming_g_sbgf = g_pming - g_sbgf
    else abs_g_pming_g_sbgf = -(g_pming - g_sbgf)
    end.

prop p_abs_g_sbfm_g_sbgf := if [ g_sbfm - g_sbgf >= 0 ]
    then abs_g_sbfm_g_sbgf = g_sbfm - g_sbgf
    else abs_g_sbfm_g_sbgf = -(g_sbfm - g_sbgf)
    end.

prop p_abs_maxf_minf_maxg_ming :=

```

```

if [ maxreducf + minreducf - maxreducg - minreducg >= 0 ]
then abs_maxf_minf_maxg_ming =
maxreducf + minreducf - maxreducg - minreducg
else abs_maxf_minf_maxg_ming =
-(maxreducf + minreducf - maxreducg - minreducg)
end.

```

```

sat ~[ [hcard & constants_ax & hC & hnp &
uboundmaxf & uboundmaxg & lboundminf & lboundming &
same_bound_f_g & same_bound_g_f &
      hbx_pmaxf & hbx_pmaxg & hbx_pminf &
      hbx_pming & hbx_sbf & hbx_sbgf &
hby1_pmaxf_pmaxg & hby1_pmaxf_pminf & hby1_pmaxf_pming &
hby1_pmaxf_sbf & hby1_pmaxf_sbgf &
hby1_pmaxg_pminf & hby1_pmaxg_pming &
      hby1_pmaxg_sbf & hby1_pmaxg_sbgf &
hby1_pminf_pming & hby1_pminf_sbf & hby1_pminf_sbgf &
      hby1_pming_sbf & hby1_pming_sbgf & hby1_sbf_sbgf &
hby2_pmaxf_pmaxg & hby2_pmaxf_pminf & hby2_pmaxf_pming &
hby2_pmaxf_sbf & hby2_pmaxf_sbgf &
hby2_pmaxg_pminf & hby2_pmaxg_pming &
      hby2_pmaxg_sbf & hby2_pmaxg_sbgf &
hby2_pminf_pming & hby2_pminf_sbf & hby2_pminf_sbgf &
      hby2_pming_sbf & hby2_pming_sbgf & hby2_sbf_sbgf &
p_abs_f_pmaxf_g_pmaxf & p_abs_f_pmaxg_g_pmaxg &
p_abs_f_pminf_g_pminf & p_abs_f_pming_g_pming &
p_abs_f_sbf_g_sbf & p_abs_f_sbgf_g_sbgf &
p_abs_f_pmaxf_f_pmaxg & p_abs_f_pmaxf_f_pminf &
p_abs_f_pmaxf_f_pming & p_abs_f_pmaxf_f_sbf &
p_abs_f_pmaxf_f_sbgf & p_abs_f_pmaxg_f_pminf &
p_abs_f_pmaxg_f_pming & p_abs_f_pmaxg_f_sbf &
p_abs_f_pmaxg_f_sbgf & p_abs_f_pminf_f_pming &
p_abs_f_pminf_f_sbf & p_abs_f_pminf_f_sbgf &
p_abs_f_pming_f_sbf & p_abs_f_pming_f_sbgf &
p_abs_f_sbf_f_sbgf & p_abs_g_pmaxf_g_pmaxg &
p_abs_g_pmaxf_g_pminf & p_abs_g_pmaxf_g_pming &
p_abs_g_pmaxf_g_sbf & p_abs_g_pmaxf_g_sbgf &
p_abs_g_pmaxg_g_pminf & p_abs_g_pmaxg_g_pming &
p_abs_g_pmaxg_g_sbf & p_abs_g_pmaxg_g_sbgf &
p_abs_g_pminf_g_pming & p_abs_g_pminf_g_sbf &
p_abs_g_pminf_g_sbgf & p_abs_g_pming_g_sbf &
p_abs_g_pming_g_sbgf & p_abs_g_sbf_g_sbgf &
p_abs_maxf_minf_maxg_ming

```

```
] => abs_maxf_minf_maxg_ming <= y + 2 * x ].
```

A.3 Accuracy Preservation property

The proof of this property was successful in both tools. Even in CVC-lite the proof was find without the need of removing the quantifiers.

File `accur_pres.cvc`:

```
%Version with all the quantifiers
% It work :)

SETPROC : TYPE;
PROC : TYPE;

np, khl: INT;
maxreduc: (PROC -> REAL, SETPROC) -> REAL;
minreduc: (PROC -> REAL, SETPROC) -> REAL;

y : REAL;

f : PROC -> REAL;
g : PROC -> REAL;
q : PROC;

PR, C : SETPROC;

card : SETPROC -> INT;
INCL : (SETPROC,SETPROC) -> BOOLEAN;
INSET : (PROC, SETPROC) -> BOOLEAN;

abs: REAL -> REAL = LAMBDA (x:REAL): IF x>=0 THEN x ELSE (-x) ENDIF;

constants_ax: BOOLEAN = 2*khl < np AND khl >= 0;

min_le_max : BOOLEAN = minreduc( f, PR) <= maxreduc( f, PR);

uboundmaxf: BOOLEAN = FORALL (C : SETPROC):
    INCL(C,PR) AND np <= card(C) + khl
    => EXISTS (i:PROC): INSET(i,C) AND maxreduc( f, PR) <= f(i);

lboundminf: BOOLEAN = FORALL (C : SETPROC):
    INCL(C,PR) AND np <= card(C) + khl
    => EXISTS (i:PROC): INSET(i,C) AND minreduc( f, PR) >= f(i);
```

```

hC : BOOLEAN = INCL(C,PR);
hnp : BOOLEAN = np <= card(C) + khl;
hqC : BOOLEAN = INSET(q,C);

hby : BOOLEAN = FORALL (C : SETPROC, l:PROC): INSET(l,C) =>
                FORALL (m:PROC): INSET(m,C) => abs(f(l) - f(m)) <= y;

ASSERT(hC AND hnp AND hqC AND min_le_max AND
uboundmaxf AND
lboundminf AND
        hby);

QUERY( abs(maxreduc(f,PR) + minreduc(f,PR)
        - 2 * f(q)) <=
        2 * y );

DUMP_PROOF; % 2192 lines

File accur_pres.ics:

% It work :)

sig np: int.
sig khl: int.

sig maxreducf: real.
sig minreducf: real.
sig maxreducg: real.
sig minreducg: real.

sig y: real.

sig f_pmaxf: real.
sig f_pminf: real.
sig f_q: real.

% f : PROC -> REAL;
% g : PROC -> REAL;
% pmaxf, pmaxg, pminf, pming : PROC;

```

```

% sbfg, sbgf: PROC;

% PR, C : SETPROC;

sig card_C : int.

% INCL : (SETPROC,SETPROC) -> BOOLEAN;
% INSET : (PROC, SETPROC) -> BOOLEAN;

prop constants_ax := 2*khl < np & khl >= 0.
prop hC := INCL_C_PR.
prop hnp := np <= card_C + khl.
prop hqC := INSET_q_C.
prop min_le_max := minreducf <= maxreducf.

prop uboundmaxf :=
  [INCL_C_PR & np <= card_C + khl]
  => [INSET_pmaxf_C & maxreducf <= f_pmaxf].

prop lboundminf :=
  [INCL_C_PR & np <= card_C + khl]
  => [INSET_pminf_C & minreducf >= f_pminf].

prop hby_pmaxf_pminf := INSET_pmaxf_C =>
  [INSET_pminf_C => abs_f_pmaxf_f_pminf <= y].
prop hby_pmaxf_q := INSET_pmaxf_C =>
  [INSET_q_C => abs_f_pmaxf_f_q <= y].

prop hby_pminf_q := INSET_pminf_C =>
  [INSET_q_C => abs_f_pminf_f_q <= y].

% abs: REAL -> REAL = LAMBDA (x:REAL): IF x>=0 THEN x ELSE (-x) ENDIF;

prop p_abs_f_pmaxf_f_pminf := if [ f_pmaxf - f_pminf >= 0 ]
  then abs_f_pmaxf_f_pminf = f_pmaxf - f_pminf
  else abs_f_pmaxf_f_pminf = -(f_pmaxf - f_pminf)
  end.
prop p_abs_f_pmaxf_f_q := if [ f_pmaxf - f_q >= 0 ]
  then abs_f_pmaxf_f_q = f_pmaxf - f_q
  else abs_f_pmaxf_f_q = -(f_pmaxf - f_q)
  end.
prop p_abs_f_pminf_f_q := if [ f_pminf - f_q >= 0 ]
  then abs_f_pminf_f_q = f_pminf - f_q

```

```

else abs_f_pminf_f_q = -(f_pminf - f_q)
end.

prop p_abs_maxf_minf_2_f_q :=
if [ maxreducf + minreducf - 2 * f_q >= 0 ]
then abs_maxf_minf_2_f_q =
maxreducf + minreducf - 2 * f_q
else abs_maxf_minf_2_f_q =
-(maxreducf + minreducf - 2 * f_q)
end.

sat ~[ [ hC & hnp & hqC & min_le_max &
uboundmaxf & lboundminf &
hby_pmaxf_pminf & hby_pmaxf_q &
hby_pminf_q &
p_abs_f_pmaxf_f_pminf & p_abs_f_pmaxf_f_q &
p_abs_f_pminf_f_q &
p_abs_maxf_minf_2_f_q
] => abs_maxf_minf_2_f_q <= 2 * y ].

```

References

- [1] D. Barsotti, L. P. Nieto, and A. Tiu. Verification of clock synchronization algorithms: Experiments on a combination of deductive tools. In *Proceedings of AVOCS 2005*, volume 145 of *ENTCS*, pages 63–68. Elsevier Science B. V., 2005. Available by WWW from <http://www.cs.famaf.unc.edu.ar/~damian/publications/clock.pdf>.
- [2] CVC Lite home page. <http://verify.stanford.edu/CVCL/>, 2006.
- [3] ICS: Integrated Canonizer and Solver home page. <http://ics.csl.sri.com/>, 2006.
- [4] L. Lamport and P. M. Melliar-Smith. Synchronizing clocks in the presence of faults. *J. ACM*, 32(1):52–78, 1985.
- [5] J. Lundelius and N. Lynch. A new fault-tolerant algorithm for clock synchronization. In *PODC '84: Proceedings of the third annual ACM symposium on Principles of distributed computing*, pages 75–88, New York, NY, USA, 1984. ACM Press.

- [6] P. S. Miner. Verification of fault-tolerant clock synchronization systems. NASA Technical Paper 3349, NASA Langley Research Center, November 1993.
- [7] F. B. Schneider. Understanding protocols for Byzantine clock synchronization. Technical Report TR 87-859, Cornell University, Dept. of Computer Science, Upson Hall, Ithaca, NY 14853, 1987.
- [8] N. Shankar. Mechanical verification of a generalized protocol for byzantine fault tolerant clock synchronization. In J. Vytupil, editor, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 571 of *Lecture Notes in Computer Science*, pages 217-236, Nijmegen, The Netherlands, jan 1992. Springer-Verlag.