

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	6

1.4	Accuracy Preservation property	6
1.4.1	Main theorem	7
2	Fault-tolerant Midpoint algorithm	7
2.1	Model of the system	7
2.1.1	Types in the formalization	7
2.1.2	Some constants	8
2.1.3	Convergence function	8
2.2	Translation Invariance property.	9
2.2.1	Auxiliary lemmas	9
2.2.2	Main theorem	11
2.3	Precision Enhancement property	11
2.3.1	Auxiliary lemmas	11
2.3.2	Main theorem	13
2.4	Accuracy Preservation property	13
A	CVC-lite and ICS proofs	13
A.1	Lemma <code>abs_distrib_div</code>	13
A.2	Bound for Precision Enhancement property	15
A.3	Accuracy Preservation property	31

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'*:

$$\begin{aligned}
& (\sum l \in \{..<np'\}. f l X (\lambda y. f y + x) p l) = \\
& \quad (\sum l \in \{..<np'\}. f l X f p l) + \text{real } np' * x \\
& \langle \text{proof} \rangle
\end{aligned}$$

theorem *trans-inv*:

$$\forall p f x . \text{cfni } p (\lambda y. f y + x) = \text{cfni } p f + x \\
\langle \text{proof} \rangle$$

1.3 Precision Enhancement property

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

1.3.1 Auxiliary lemmas

lemma *finitC*:

$$C \subseteq PR \implies \text{finite } C \\
\langle \text{proof} \rangle$$

lemma *finitnpC*:

$$\text{finite } (PR - C) \\
\langle \text{proof} \rangle$$

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

lemma *sum-abs-triangle-ineq*:

$$\text{finite } S \implies \\
|\sum l \in S. (f :: 'a \Rightarrow 'b :: \text{ordered-idom}) l| \leq (\sum l \in S. |f l|) \\
(\text{is } \dots \implies ?P S) \\
\langle \text{proof} \rangle$$

lemma *sum-le*:

$$[[\text{finite } S ; \forall r \in S. f r \leq b]] \\
\implies \\
(\sum l \in S. f l) \leq \text{real } (\text{card } S) * b \\
(\text{is } [[\text{finite } S ; \forall r \in S. f r \leq b]] \implies ?P S) \\
\langle \text{proof} \rangle$$

lemma *sum-np-eq*:

assumes

$$hC: C \subseteq PR$$

shows

$$(\sum l \in \{..<np\}. f l) = (\sum l \in C. f l) + (\sum l \in (\{..<np\} - C). f l) \\
\langle \text{proof} \rangle$$

lemma *abs-sum-np-ineq*:

assumes

$$hC: C \subseteq PR$$

shows

$$|(\sum l \in \{..<np\}. (f :: \text{nat} \Rightarrow \text{real}) l)| \leq$$

$$(\sum l \in C. |f l|) + (\sum l \in (\{..<np\} - C). |f l|)$$

(**is** ?abs-sum <= ?sumC + ?sumnpC)

<proof>

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>

lemma *fiX-lbound*:

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

<proof>

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

<proof>

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>

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

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

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

<proof>

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⟩

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⟩

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

$(real (card C) * (x + (if (y \leq \Delta) then 0 else y))) +$
 $real (card \{..\lt np\} - C) * (2 * \Delta + x + y) / real np$
 (**is** $|?dif-div-np| \leq ?B$)

⟨proof⟩

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*:

$(\sum l \in \{..\lt n::nat\}. f l) + q * real n =$
 $(\sum l \in \{..\lt n\}. f l + q)$
 (**is** $?Sl n = ?Sr n$)

⟨proof⟩

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
 $|fX f p r - f q| \leq x$
 $\langle proof \rangle$

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
 $|fX f p r - f q| \leq x + \Delta$
 $\langle proof \rangle$

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 $|cfni p f - f q| \leq$
 $(real (card C) * x + real (card \{..<np\} - C)) * (x + \Delta)) /$
 $real np$
 $(is ?abs1 \leq (?bC + ?bnpC) / real np)$
 $\langle proof \rangle$
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 \Rightarrow Clocktime) \Rightarrow process set \Rightarrow Clocktime set **where**
reduce f P = f ‘ (P - (kmax f P \cup kmin f P))

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

definition

cfnl :: process \Rightarrow (process \Rightarrow Clocktime) \Rightarrow 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 \Rightarrow ('b::linorder)
assumes fin: finite S
shows S \neq {} $\implies \exists m \in S. \forall s \in S. f s \leq f m$
 <proof>

lemma *ex-Minf*:

fixes S **and** f :: 'a \Rightarrow ('b::linorder)
assumes fin: finite S
shows S \neq {} $\implies \exists m \in S. \forall s \in S. f m \leq f s$
 <proof>

This trivial lemma is needed by the next two.

lemma *khl-bound*: khl < np

<proof>

The next two lemmas prove that the 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

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

$(\forall i \in (kmax\ f\ PR). \forall j \in PR - (kmax\ f\ PR). f\ j \leq f\ i)$
 <proof>

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>

The next two lemmas are trivial from the previous ones

lemma *finite-kmax*:

finite $(kmax\ f\ PR)$

<proof>

lemma *finite-kmin*:

finite $(kmin\ f\ PR)$

<proof>

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

lemma *reduce-not-empty*:

reduce $f\ PR \neq \{\}$

<proof>

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

lemma *reduce-shift*:

fixes $f :: nat \Rightarrow Clocktime$

shows

$f\ ' (PR - (kmax\ f\ PR \cup kmin\ f\ PR)) =$
 $f\ ' (PR - (kmax\ (\lambda\ p. f\ p + c)\ PR \cup kmin\ (\lambda\ p. f\ p + c)\ PR))$

<proof>

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>

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>

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>

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: $card\ A + k \leq card\ B + card\ C$

shows $k \leq card\ (B \cap C)$

<proof>

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: $card\ A < card\ B + card\ C$

shows

$B \cap C \neq \{\}$

<proof>

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*:

$\forall c \in reduce\ f\ PR. \exists i \in PR - (kmax\ f\ PR \cup kmin\ f\ PR). f\ i = c$

<proof>

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

lemma *finite-reduce*:
finite (*reduce f PR*)
 ⟨*proof*⟩

lemma *kmax-ge*:
 $\forall i \in (kmax\ f\ PR). \forall r \in (reduce\ f\ PR). r \leq f\ i$
 ⟨*proof*⟩

lemma *kmin-le*:
 $\forall i \in (kmin\ f\ PR). \forall r \in (reduce\ f\ PR). f\ i \leq r$
 ⟨*proof*⟩

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*⟩

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**
 $hCk: np \leq card\ C + khl$
shows
 $\exists i \in C. Max\ (reduce\ f\ PR) \leq f\ i$
 ⟨*proof*⟩

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*⟩

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*⟩

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*)

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*)

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 were 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_sbfg : BOOLEAN = INSET(sbfg,C) => abs(f(sbfg) - g(sbfg)) <= x;
hbx_sbgf : BOOLEAN = INSET(sbgf,C) => 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;

```

```

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_sbfq AND hbx_sbgf 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(sbfg)
    AND maxreduc( g, PR) >= g(sbfg));

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_sbfg : BOOLEAN = abs(f(sbfg) - g(sbfg)) <= 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_sbfg : BOOLEAN = (abs(f(pmaxf) - f(sbfg)) <= 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_sbf g : BOOLEAN = (abs(f(pmaxg) - f(sbf g)) <= y);
hby1_pmaxg_sbf g : BOOLEAN = (abs(f(pmaxg) - f(sbf g)) <= y);

hby1_pminf_pming : BOOLEAN = (abs(f(pminf) - f(pming)) <= y);
hby1_pminf_sbf g : BOOLEAN = (abs(f(pminf) - f(sbf g)) <= y);
hby1_pminf_sbf g : BOOLEAN = (abs(f(pminf) - f(sbf g)) <= y);

hby1_pming_sbf g : BOOLEAN = (abs(f(pming) - f(sbf g)) <= y);
hby1_pming_sbf g : BOOLEAN = (abs(f(pming) - f(sbf g)) <= y);

hby1_sbf g_sbf g : BOOLEAN = (abs(f(sbf g) - f(sbf g)) <= 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_sbf g : BOOLEAN = (abs(g(pmaxf) - g(sbf g)) <= y);
hby2_pmaxf_sbf g : BOOLEAN = (abs(g(pmaxf) - g(sbf g)) <= y);

hby2_pmaxg_pminf : BOOLEAN = (abs(g(pmaxg) - g(pminf)) <= y);
hby2_pmaxg_pming : BOOLEAN = (abs(g(pmaxg) - g(pming)) <= y);
hby2_pmaxg_sbf g : BOOLEAN = (abs(g(pmaxg) - g(sbf g)) <= y);
hby2_pmaxg_sbf g : BOOLEAN = (abs(g(pmaxg) - g(sbf g)) <= y);

hby2_pminf_pming : BOOLEAN = (abs(g(pminf) - g(pming)) <= y);
hby2_pminf_sbf g : BOOLEAN = (abs(g(pminf) - g(sbf g)) <= y);
hby2_pminf_sbf g : BOOLEAN = (abs(g(pminf) - g(sbf g)) <= y);

hby2_pming_sbf g : BOOLEAN = (abs(g(pming) - g(sbf g)) <= y);
hby2_pming_sbf g : BOOLEAN = (abs(g(pming) - g(sbf g)) <= y);

hby2_sbf g_sbf g : BOOLEAN = (abs(g(sbf g) - g(sbf g)) <= 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_sbf g AND hbx_sbf g 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 AND
hby2_pmaxf_sbf AND hby2_pmaxf_sbf AND
hby2_pmaxg_pminf AND hby2_pmaxg_pming AND
  hby2_pmaxg_sbf AND hby2_pmaxg_sbf AND
hby2_pminf_pming AND hby2_pminf_sbf AND hby2_pminf_sbf AND
  hby2_pming_sbf AND hby2_pming_sbf AND hby2_sbf_sbf);

```

```

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_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;

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_sbfq AND hbx_sbgf AND
hby1_pmaxf_pmaxg AND hby1_pmaxf_pminf AND hby1_pmaxf_pming AND
hby1_pmaxf_sbfq AND hby1_pmaxf_sbgf AND
hby1_pmaxg_pminf AND hby1_pmaxg_pming AND
    hby1_pmaxg_sbfq AND hby1_pmaxg_sbgf AND
hby1_pminf_pming AND hby1_pminf_sbfq AND hby1_pminf_sbgf AND
    hby1_pming_sbfq AND hby1_pming_sbgf AND hby1_sbfq_sbgf 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;
%CONTEREXAMPLE;

```

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_sbfq: real.

```

```

sig f_sbgf: real.

sig g_pmaxf: real.
sig g_pmaxg: real.
sig g_pminf: real.
sig g_pming: real.
sig g_sbgf: real.
sig g_sbgf: 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 & 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_sbgf_C & minreducf <= f_sbgf
& maxreducg >= g_sbgf].

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

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_sbgf := INSET_sbgf_C => abs_f_sbgf_g_sbgf <= x.
prop hbx_sbgf := 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_sbgf := INSET_pmaxf_C =>
  [INSET_sbgf_C => abs_f_pmaxf_f_sbgf <= y].
prop hby1_pmaxf_sbgf := 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_sbgf := INSET_pmaxg_C =>
  [INSET_sbgf_C => abs_f_pmaxg_f_sbgf <= 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_sbgf := INSET_pminf_C =>
  [INSET_sbgf_C => abs_f_pminf_f_sbgf <= y].
prop hby1_pminf_sbgf := INSET_pminf_C =>
  [INSET_sbgf_C => abs_f_pminf_f_sbgf <= y].

prop hby1_pming_sbgf := INSET_pming_C =>

```

```

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

prop hby1_sbgf_sbgf := INSET_sbgf_C =>
[INSET_sbgf_C => abs_f_sbgf_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_sbgf := INSET_pmaxf_C =>
[INSET_sbgf_C => abs_g_pmaxf_g_sbgf <= 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_sbgf := INSET_pmaxg_C =>
[INSET_sbgf_C => abs_g_pmaxg_g_sbgf <= 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_sbgf := INSET_pminf_C =>
[INSET_sbgf_C => abs_g_pminf_g_sbgf <= y].
prop hby2_pminf_sbgf := INSET_pminf_C =>
[INSET_sbgf_C => abs_g_pminf_g_sbgf <= y].

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

prop hby2_sbgf_sbgf := INSET_sbgf_C =>
[INSET_sbgf_C => abs_g_sbgf_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_sbf_g_sbf := if [ f_sbf - g_sbf >= 0 ]
    then abs_f_sbf_g_sbf = f_sbf - g_sbf
    else abs_f_sbf_g_sbf = -(f_sbf - g_sbf)
    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_sbf := if [ f_pmaxf - f_sbf >= 0 ]
    then abs_f_pmaxf_f_sbf = f_pmaxf - f_sbf
    else abs_f_pmaxf_f_sbf = -(f_pmaxf - f_sbf)
    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.

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_sbfg := if [ f_pming - f_sbfg >= 0 ]
  then abs_f_pming_f_sbfg = f_pming - f_sbfg
  else abs_f_pming_f_sbfg = -(f_pming - f_sbfg)
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_sbf_g_f_sbf_g := if [ f_sbf_g - f_sbf_g >= 0 ]
    then abs_f_sbf_g_f_sbf_g = f_sbf_g - f_sbf_g
    else abs_f_sbf_g_f_sbf_g = -(f_sbf_g - f_sbf_g)
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_sbf_g := if [ g_pmaxf - g_sbf_g >= 0 ]
    then abs_g_pmaxf_g_sbf_g = g_pmaxf - g_sbf_g
    else abs_g_pmaxf_g_sbf_g = -(g_pmaxf - g_sbf_g)
end.

prop p_abs_g_pmaxf_g_sbf_g := if [ g_pmaxf - g_sbf_g >= 0 ]
    then abs_g_pmaxf_g_sbf_g = g_pmaxf - g_sbf_g
    else abs_g_pmaxf_g_sbf_g = -(g_pmaxf - g_sbf_g)
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_sbf_g := if [ g_pmaxg - g_sbf_g >= 0 ]
    then abs_g_pmaxg_g_sbf_g = g_pmaxg - g_sbf_g
    else abs_g_pmaxg_g_sbf_g = -(g_pmaxg - g_sbf_g)
end.

prop p_abs_g_pmaxg_g_sbf_g := if [ g_pmaxg - g_sbf_g >= 0 ]
    then abs_g_pmaxg_g_sbf_g = g_pmaxg - g_sbf_g
    else abs_g_pmaxg_g_sbf_g = -(g_pmaxg - g_sbf_g)
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_sbf_g := if [ g_pminf - g_sbf_g >= 0 ]
    then abs_g_pminf_g_sbf_g = g_pminf - g_sbf_g
    else abs_g_pminf_g_sbf_g = -(g_pminf - g_sbf_g)
    end.

prop p_abs_g_pminf_g_sbf_g := if [ g_pminf - g_sbf_g >= 0 ]
    then abs_g_pminf_g_sbf_g = g_pminf - g_sbf_g
    else abs_g_pminf_g_sbf_g = -(g_pminf - g_sbf_g)
    end.

prop p_abs_g_pming_g_sbf_g := if [ g_pming - g_sbf_g >= 0 ]
    then abs_g_pming_g_sbf_g = g_pming - g_sbf_g
    else abs_g_pming_g_sbf_g = -(g_pming - g_sbf_g)
    end.

prop p_abs_g_pming_g_sbf_g := if [ g_pming - g_sbf_g >= 0 ]
    then abs_g_pming_g_sbf_g = g_pming - g_sbf_g
    else abs_g_pming_g_sbf_g = -(g_pming - g_sbf_g)
    end.

prop p_abs_g_sbf_g_sbf_g := if [ g_sbf_g - g_sbf_g >= 0 ]
    then abs_g_sbf_g_sbf_g = g_sbf_g - g_sbf_g
    else abs_g_sbf_g_sbf_g = -(g_sbf_g - g_sbf_g)
    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_g & hbx_sbf_g &
hby1_pmaxf_pmaxg & hby1_pmaxf_pminf & hby1_pmaxf_pming &
hby1_pmaxf_sbf_g & hby1_pmaxf_sbf_g &

```

```

hby1_pmaxg_pminf & hby1_pmaxg_pming &
  hby1_pmaxg_sbf & hby1_pmaxg_sbf &
hby1_pminf_pming & hby1_pminf_sbf & hby1_pminf_sbf &
  hby1_pming_sbf & hby1_pming_sbf & hby1_sbf_sbf &
hby2_pmaxf_pmaxg & hby2_pmaxf_pminf & hby2_pmaxf_pming &
hby2_pmaxf_sbf & hby2_pmaxf_sbf &
hby2_pmaxg_pminf & hby2_pmaxg_pming &
  hby2_pmaxg_sbf & hby2_pmaxg_sbf &
hby2_pminf_pming & hby2_pminf_sbf & hby2_pminf_sbf &
  hby2_pming_sbf & hby2_pming_sbf & hby2_sbf_sbf &
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_sbf_g_sbf &
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_sbf & p_abs_f_pmaxg_f_pminf &
p_abs_f_pmaxg_f_pming & p_abs_f_pmaxg_f_sbf &
p_abs_f_pmaxg_f_sbf & p_abs_f_pminf_f_pming &
p_abs_f_pminf_f_sbf & p_abs_f_pminf_f_sbf &
p_abs_f_pming_f_sbf & p_abs_f_pming_f_sbf &
p_abs_f_sbf_f_sbf & 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_sbf &
p_abs_g_pmaxg_g_pminf & p_abs_g_pmaxg_g_pming &
p_abs_g_pmaxg_g_sbf & p_abs_g_pmaxg_g_sbf &
p_abs_g_pminf_g_pming & p_abs_g_pminf_g_sbf &
p_abs_g_pminf_g_sbf & p_abs_g_pming_g_sbf &
p_abs_g_pming_g_sbf & p_abs_g_sbf_g_sbf &
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 found 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. Vytopil, 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.