Instances of Schneider's generalized protocol of clock synchronization.

Damian Barsotti

March 17, 2025

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://isa-afp.org/entries/GenClock.shtml).

Contents

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.

type-synonym *process* = *nat*
type-synonym *time* = *real* — real time
type-synonym *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 cero in the definition of the convergence function).

axiomatization
np :: *nat* — Number of processes **and**
Δ :: *Clocktime* — Fix value to discard processes **where**
constants-ax: *0* <= Δ ∧ *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* ⇒ *Clocktime*), *process*, *process*] ⇒ *Clocktime* **where**
fiX f p l = (*if* |*f p* − *f l*| <= Δ *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* ⇒ *Clocktime*)] ⇒ *Clocktime* **where**
cfni p f = (∑ *l*∈{..<*np*}. *fiX f p l*) / (*real np*)

1.2 Translation Invariance property.

We first need to prove this auxiliary lemma.

lemma *trans-inv'*:

$(\sum\ l{\in}\{..{<}np'\}.\ fiX\ (\lambda\ y.\ f\ y\ +\ x)\ p\ l) =$
$\quad (\sum\ l{\in}\{..{<}np'\}.\ fiX\ f\ p\ l)\ +\ real\ np' * x$
⟨*proof*⟩

theorem *trans-inv*:
$\forall\ p\ f\ x\ .\ cfni\ p\ (\lambda\ y.\ f\ y\ +\ x) = cfni\ p\ f\ +\ x$
⟨*proof*⟩

1.3 Precision Enhancement property

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

1.3.1 Auxiliary lemmas

lemma *finitC*:
$C \subseteq PR \Longrightarrow finite\ C$
⟨*proof*⟩

lemma *finitnpC*:
$finite\ (PR - C)$
⟨*proof*⟩

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

lemma *sum-abs-triangle-ineq*:
finite S $\Longrightarrow$
$|\sum l{\in}S.\ (f{::}'a \Rightarrow 'b{::}linordered\text{-}idom)\ l| <= (\sum l{\in}S.\ |f\ l|)$
(**is** ... $\Longrightarrow$ *?P S*)
⟨*proof*⟩

lemma *sum-le*:
$[\![finite\ S\ ;\ \forall\ r{\in}S.\ f\ r <= b\ ]\!]$
$\Longrightarrow$
$(\sum l{\in}S.\ f\ l) <= real\ (card\ S) * b$
(**is** $[\![\ finite\ S\ ;\ \forall\ r{\in}S.\ f\ r <= b\ ]\!] \Longrightarrow$ *?P S*)
⟨*proof*⟩

lemma *sum-np-eq*:
assumes
hC: $C \subseteq PR$
shows
$(\sum l{\in}\{..{<}np\}.\ f\ l) = (\sum l{\in}C.\ f\ l) + (\sum l{\in}(\{..{<}np\}-C).\ f\ l)$
⟨*proof*⟩

lemma *abs-sum-np-ineq*:
assumes
hC: $C \subseteq PR$
shows
$|(\sum l{\in}\{..{<}np\}.\ (f{::}nat \Rightarrow real)\ l)| <=$

$(\sum l{\in}C.\ |f\ l|) + (\sum l{\in}(\{..{<}np\}{-}C).\ |f\ l|)$
(**is** *?abs-sum* <= *?sumC* + *?sumnpC*)
⟨*proof*⟩

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

lemma *fiX-ubound*:
fiX f p l <= *f p* + Δ
⟨*proof*⟩

lemma *fiX-lbound*:
f p − Δ <= *fiX f p l*
⟨*proof*⟩

lemma *abs-fiX-bound*: |*fiX f p l* − *f p* | <= Δ
⟨*proof*⟩

lemma *abs-dif-fiX-bound*:
assumes
hbx: ∀ *l*∈*C*. |*f l* − *g l*| <= *x* **and**
hby: ∀ *l*∈*C*. ∀ *m*∈*C*. |*f l* − *f m*| <= *y* **and**
hpC: *p*∈*C* **and**
hqC: *q*∈*C*
shows
|*fiX f p r* − *fiX g q r*| <= *2* ∗ Δ + *x* + *y*
⟨*proof*⟩

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*⟩

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

$y <= \Delta \longrightarrow |fiX\ f\ p\ r - fiX\ g\ q\ r| <= x$
⟨*proof*⟩

lemma *abs-dif-fiX-bound-C*:
assumes
hbx: $\forall\ l{\in}C.\ |f\ l - g\ l| <= x$ **and**
hby1: $\forall\ l{\in}C.\ \forall\ m{\in}C.\ |f\ l - f\ m| <= y$ **and**
hby2: $\forall\ l{\in}C.\ \forall\ m{\in}C.\ |g\ l - g\ m| <= y$ **and**
hpC: $p{\in}C$ **and**
hqC: $q{\in}C$ **and**
hrC: $r{\in}C$
shows
$|fiX\ f\ p\ r - fiX\ g\ q\ r| <=$
$x + (if\ (y <= \Delta)\ then\ 0\ else\ y)$
⟨*proof*⟩

1.3.2 Main theorem

theorem *prec-enh*:
assumes
hC: $C \subseteq PR$ **and**
hbx: $\forall\ l{\in}C.\ |f\ l - g\ l| <= x$ **and**
hby1: $\forall\ l{\in}C.\ \forall\ m{\in}C.\ |f\ l - f\ m| <= y$ **and**
hby2: $\forall\ l{\in}C.\ \forall\ m{\in}C.\ |g\ l - g\ m| <= y$ **and**
hpC: $p{\in}C$ **and**
hqC: $q{\in}C$
shows $|\ cfni\ p\ f - cfni\ q\ g\ | <=$
$(real\ (card\ C) * (x + (if\ (y <= \Delta)\ then\ 0\ else\ y)) +$
$real\ (card\ (\{..<np\} - C)) * (2 * \Delta + x + y)) / real\ np$
(**is** $|\ ?dif\text{-}div\text{-}np\ | <= ?B$)
⟨*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*:
$(\sum l{\in}\{..<n::nat\}.\ f\ l) + q * real\ n =$
$(\sum l{\in}\{..<n\}.\ f\ l + q\ )$
(**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: $\forall\ l{\in}C.\ \forall\ m{\in}C.\ |f\ l - f\ m| <= x$ **and**
hpC: $p{\in}C$ **and**

hqC: $q \in C$ **and**
hrC: $r \in C$
shows
$|fiX\ f\ p\ r - f\ q| <= x$
⟨*proof*⟩

lemma *bound-aux*:
assumes
hby: $\forall\ l \in C.\ \forall\ m \in C.\ |f\ l - f\ m| <= x$ **and**
hpC: $p \in C$ **and**
hqC: $q \in C$
shows
$|fiX\ f\ p\ r - f\ q| <= x + \Delta$
⟨*proof*⟩

1.4.1 Main theorem

lemma *accur-pres*:
assumes
hC: $C \subseteq PR$ **and**
hby: $\forall\ l \in C.\ \forall\ m \in C.\ |f\ l - f\ m| <= x$ **and**
hpC: $p \in C$ **and**
hqC: $q \in C$
shows $|\ cfni\ p\ f - f\ q\ | <=$
$(real\ (card\ C) * x + real\ (card\ (\{..<np\} - C)) * (x + \Delta))/$
$real\ np$
(**is** $?abs1 <= (?bC + ?bnpC)/real\ np$)
⟨*proof*⟩

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.

type-synonym *process* = *nat*
type-synonym *time* = *real* — real time
type-synonym *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 eleccion 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**
$kmax\ f\ P = (SOME\ S.\ S \subseteq P \wedge card\ S = khl \wedge$
$(\forall\ i{\in}S.\ \forall\ j{\in}(P{-}S).\ f\ j <= f\ i))$

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

With the previus functions we define a new one *reduce*[1]. This take a function of clock readings and a set of processes and return de set of readings of the not dicarded processes. In order to define this function we use the image operator ((‘)) of Isabelle.

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

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 proves 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*
⟨*proof*⟩

lemma *ex-Minf*:
fixes *S* **and** *f* :: *'a* ⇒ (*'b*::*linorder*)
assumes *fin*: *finite S*
shows *S* ≠ {} ==> ∃ *m*∈*S*. ∀ *s* ∈ *S*. *f m* ≤ *f s*
⟨*proof*⟩

This trivial lemma is needed by the next two.

lemma *khl-bound*: *khl* < *np*
⟨*proof*⟩

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 is the only thing that we know about these functions.

lemma *kmax-prop*:
fixes *f* :: *nat* ⇒ *Clocktime*
shows
(*kmax f PR*) ⊆ *PR* ∧ *card* (*kmax f PR*) = *khl* ∧
(∀ *i*∈(*kmax f PR*). ∀ *j*∈*PR* − (*kmax f PR*). *f j* ≤ *f i*)
⟨*proof*⟩

lemma *kmin-prop*:
fixes *f* :: *nat* ⇒ *Clocktime*
shows
(*kmin f PR*) ⊆ *PR* ∧ *card* (*kmin f PR*) = *khl* ∧

$$(\forall\, i{\in}(kmin\ f\ PR).\ \forall\, j{\in}PR - (kmin\ f\ PR).\ f\ i \leq f\ j)$$
⟨*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 necesary because the definition of the convergence function use the builtin Max and Min.

lemma *reduce-not-empty*:
reduce f PR ≠ {}
⟨*proof*⟩

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

lemma *reduce-shift*:
fixes *f* :: *nat* ⇒ *Clocktime*
shows
f ‘ (*PR* − (*kmax f PR* ∪ *kmin f PR*)) =
f ‘ (*PR* − (*kmax* (λ *p*. *f p* + *c*) *PR* ∪ *kmin* (λ *p*. *f p* + *c*) *PR*))
⟨*proof*⟩

lemma *max-shift*:
fixes *f* :: *nat* ⇒ *Clocktime* **and** *S*
assumes *notEmpFin*: *S* ≠ {} *finite S*
shows
Max (*f*‘*S*) + *x* = *Max* ((λ *p*. *f p* + *x*) ‘ *S*)
⟨*proof*⟩

lemma *min-shift*:
fixes *f* :: *nat* ⇒ *Clocktime* **and** *S*
assumes *notEmpFin*: *S* ≠ {} *finite S*
shows
Min (*f*‘*S*) + *x* = *Min* ((λ *p*. *f p* + *x*) ‘ *S*)
⟨*proof*⟩

2.2.2 Main theorem

theorem *trans-inv*:
fixes *f* :: *nat* ⇒ *Clocktime*
shows
cfnl p f + *x* = *cfnl p* (λ *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 consecuence 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* <= *card B* + *card C*
shows *k* <= *card* $(B \cap C)$
⟨*proof*⟩

This lemma is a trivial consecuence 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* ∪ *kmin f PR*) by the function *f*.

lemma *exist-reduce*:
∀ *c* ∈ *reduce f PR*. ∃ *i*∈ *PR*−(*kmax f PR* ∪ *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*:
∀ *i*∈ (*kmax f PR*). ∀ *r* ∈ (*reduce f PR*). *r* <= *f i*
⟨*proof*⟩

lemma *kmin-le*:
$\forall\ i \in (kmin\ f\ PR).\ \forall\ r \in (reduce\ f\ PR).\ f\ i <= r$
⟨*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 $0 <= c$!!

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

lemma *lboundmin*:
assumes
hC: $C \subseteq PR$ **and**
hCk: $np <= card\ C + khl$
shows
$\exists\ i \in C.\ f\ i <= Min\ (reduce\ f\ PR)$
⟨*proof*⟩

lemma *same-bound*:
assumes
hC: $C \subseteq PR$ **and**
hCk: $np <= card\ C + khl$ **and**
hnk: $3 * khl < np$
shows
$\exists\ i \in C.\ Min\ (reduce\ f\ PR) <= f\ i \wedge g\ i <= Max\ (reduce\ g\ PR)$
⟨*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: $np - nF <= card\ C$ **and**
hFn: $3 * nF < np$ **and**
hFk: $nF = khl$ **and**
hbx: $\forall\ l{\in}C.\ |f\ l - g\ l| <= x$ **and**
hby1: $\forall\ l{\in}C.\ \forall\ m{\in}C.\ |f\ l - f\ m| <= y$ **and**
hby2: $\forall\ l{\in}C.\ \forall\ m{\in}C.\ |g\ l - g\ m| <= y$ **and**
hpC: $p{\in}C$ **and**
hqC: $q{\in}C$
shows $|\ cfnl\ p\ f - cfnl\ q\ g\ | <= y\ /\ 2 + x$
⟨*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: $np - nF <= card\ C$ **and**
hFk: $nF = khl$ **and**
hby: $\forall\ l{\in}C.\ \forall\ m{\in}C.\ |f\ l - f\ m| <= y$ **and**
hqC: $q{\in}C$
shows $|\ cfnl\ p\ f - f\ q\ | <= y$
⟨*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`:

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`:

File `abs_distrib_mult2.cvc`:

A.2 Bound for Precision Enhancement property

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

have $|Max\ (reduce\ f\ PR) + Min\ (reduce\ f\ PR)\ + \\ - Max\ (reduce\ g\ PR) + - Min\ (reduce\ g\ PR)| <= y + 2 * x$

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`:

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`:

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

File `bound_prec_enh7.cvc`:

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`:

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`:

File `accur_pres.ics`:

References

[1] D. Barsotti, L. P. Nieto, and A. Tiu. Verication 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.