HAL
open science

Verification of Clock Synchronization Algorithms: Experiments on a combination of deductive tools

Damian Barsotti, Leonor Prensa Nieto, Alwen Tiu

To cite this version:

Damian Barsotti, Leonor Prensa Nieto, Alwen Tiu. Verification of Clock Synchronization Algorithms: Experiments on a combination of deductive tools. Fifth International Workshop on Automated Verification of Critical Systems 2005 - AVoCS '05, Ranko Lazic, Rajagopal Nagarajan, Nikolaos Papanikolaou, Sep 2005, Coventry/UK. inria-00000638

HAL Id: inria-00000638

https://inria.hal.science/inria-00000638v1

Submitted on 10 Nov 2005

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Verification of Clock Synchronization Algorithms: Experiments on a combination of deductive tools

Damian Barsotti, Leonor Prensa Nieto and Alwen Tiu [1]

LORIA, France

Abstract

We report on an experiment in combining the theorem prover Isabelle with automatic first-order arithmetic provers to increase automation on the verification of distributed protocols. As a case study for the experiment we verify several averaging clock synchronization algorithms. We present a formalization of Schneider's generalized clock synchronization protocol [15] in Isabelle/HOL. Then, we verify that the convergence functions used in two clock synchronization algorithms, namely, the Interactive Convergence Algorithm (ICA) of Lamport and Melliar-Smith [10] and the Fault-tolerant Midpoint algorithm of Lundelius-Lynch [11], satisfy Schneider's general conditions for correctness. The proofs are completely formalized in Isabelle/HOL. We identify the parts of the proofs which are not fully automatically proven by Isabelle built-in tactics and show that these proofs can be handled by automatic first-order provers with support for arithmetic like ICS and CVC Lite.

Key words: Theorem proving, verification, clock synchronization.

1 Introduction

Achieving a high degree of automation on the verification of critical systems and in particular of distributed protocols, has been recognized as a necessity due to the complexity and the size of the verification tasks. Such verification tasks are best carried out using theorem provers that provide a powerful specification language, in order to obtain a formal description of protocols that is close to their real implementation. However, it is unrealistic in general to expect general-purpose theorem provers like Isabelle [9] to include sophisticated decision procedures needed for various verification tasks. On the other hand, there exist many specialized efficient automated provers that handle specific verification tasks, e.g., model checkers, SAT solvers, etc. It is then natural

[1] Email: {damian.barsotti,leonor.prensa,alwen.tiu}@loria.fr

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

to ask whether it is feasible to use expressive theorem provers like Isabelle as the specification language, and the specialized automated provers for handling specific parts of the verification tasks. For the domain of real-time distributed protocols, this might be the case, since verification of these protocols typically involves a number of arithmetic subproblems that can possibly be handled by automatic solvers.

We report on an experiment on combining automatic theorem provers with the interactive theorem prover Isabelle/HOL. We use Isabelle/HOL as the main specification language, and use other automated provers such as ICS [8] and CVC Lite [2] to improve the automation of proof development. The current paper focusses on a particular case study, that is, verification of clock synchronization protocols, to gain insights into the real usefulness and the feasibility of such a combination of tools. For the long term project, we consider building a verification framework for distributed protocols based on Isabelle/HOL. Fault-tolerant clock synchronization is an excellent example of a problem that requires both reasoning in higher-order logic and arithmetics. A large part of the proof involves linear integer and rational inequalities and equalities. Our experience shows that many of such lemmas cannot be proved by Isabelle's automatic tactics but are solved by ICS and/or CVC Lite.

Of course, the idea of combination of deductive tools is not new and there have been several existing works along this line, e.g., the Omega prover [18], the CALIFE project [20], the SAL framework [3] and the PROSPER project [4]. Our approach differs from these projects in the specification language used (Isabelle/HOL) and the scope of our project. We aim at providing a framework for specification and verification of distributed protocols, since these protocols are typically simple but their properties are challenging to verify formally. We also intend to have a strongly consistent verification framework, in the sense that, we shall allow the possibility of extracting explicit proof objects from the verification. This motivates our choice of Isabelle/HOL as the main specification language, since in Isabelle one can generate proof objects (based on a natural deduction proof system) which can then be verified independently. Moreover, Isabelle comes with a high-level proof language Isar which resembles the usual style of proofs found in mathematics, and hence improves readability and maintainability of proof scripts.

We note that although the work presented in this paper is aimed at combination of deductive tools, a substantial contribution of the paper is the formalized clock synchronization protocol of Schneider [15], and some parts of the ICA [10] and the Lundelius-Lynch [11] protocols. Verification of these protocols has been previously carried out in EHDM [17] and PVS [12,16], but to our knowledge, ours are the first ones done in Isabelle, from which complete formal proof objects can be produced and verified independently. These formalized proofs can serve as the basis for verification of more concrete clock synchronization protocols, such as the ones used in the FlexRay protocol [6] (for drive-by-wire application in automotive industries). There has indeed

been on-going work in the verification of FlexRay using Isabelle/HOL [2] which is complementary to our work.

The rest of the paper is organized as follows. In Section 2, we give an overview of the Isabelle/HOL and Isar systems. Section 3 gives an overview of the clock synchronization problem and the formalization of Schneider's generalized protocol for clock synchronization. Section 4 presents the formalization of two particular clock synchronization algorithms as instances of Schneider's generalized protocol: the ICA protocol and the Lundelius-Lynch protocol. Section 5 shows how parts of this formalization can be proved automatically using the ICS and CVC Lite tools. Section 6 discusses future work.

2 Isabelle/HOL and Isar

Isabelle [9] is a generic interactive proof assistant. Generic means that it can be instantiated with different object logics. We use Isabelle/HOL which is the instance for higher-order logic. Interactive means that proving theorems requires guidance from the user. The main drawback of such proof systems is the expertise needed to perform proofs with a reasonable amount of effort. Isabelle provides some tools called tactics that are able to automatically prove some parts of the proofs. In particular, the *classical reasoner* implements a tableau based prover for predicate logic and sets that can perform long chains of reasoning steps. The *simplifier* can reason with and about equations. The tactic *arith* proves linear arithmetic facts automatically. More ambitious tactics like *auto* or *force* combine the precedent ones and are able to prove, or considerably simplify, complicated theorems.

Isabelle is mainly used for the formalization of mathematical proofs and in particular formal verification, which includes proving the correctness of computer hardware or software and proving properties of computer languages and protocols.

Isar is an extension of Isabelle with structured proofs very similar to those used in mathematics texts. With Isar, users are able to produce proof scripts naturally understandable for both humans and computers. We can consider Isabelle's original tactic proof style analogous to assembly language and Isar proof scripts as high-level structured commented programs. The latter is suitable for communication and easier to maintain.

We introduce some basic notions and notations from Isabelle and then show briefly what a typical Isar proof looks like.

Isabelle's meta-logic comes with a type of *propositions* with implication $\Longrightarrow$ and a universal quantifier $\bigwedge$ for expressing inference rules and generality. The notation $[\![A_1; \ldots; A_n]\!] \Longrightarrow A$ represents an implication with assumptions $A_1, \ldots, A_n$ and conclusion A.

[2] Christian Kuehnel (Verisoft), private communication.

Isabelle terms are simply typed using Church's type system. Function types are written $\tau_1 \Rightarrow \tau_2$. Constants are declared with **consts** followed by their name and type, separated by '::'. Non-recursive definitions are declared by the keyword **constdefs**. The introduced constant and its definition are separated by '≡'.

The notation and semantics of Isar are self-explanatory. We only explain briefly the basic structure and some features that appear in this paper. A typical Isar proof skeleton would be

theorem *example*:
  assumes *assm*: *formula-0*
shows *formula-n+1*
proof
assume *formula-0*
have *formula-1* **by** *simp*
  ⋮
have *formula-n* **by** *blast*
show *formula-n+1* ..
qed

It proves *formula-0* $\Longrightarrow$ *formula-n+1*. Stating theorems using the keyword **assumes** allows naming of assumptions. The **have**s in between are the intermediate results derived from the assumptions and/or previous results. The last **show** establishes the conclusion of the theorem. The keyword **proof** announces the beginning of the proof but also tries to select an introduction rule from a predefined list of rules that suits the goal. This first step can be avoided by writing **proof**−. The ".." abbreviates a proof done by application of a single rule from a predefined set of introduction rules. When we want to use an elimination rule we can explicity indicate the premise to be eliminated using

from *facts (***show** | **have***) proposition* **by** *some-proof-method*

Other features that appear in this paper are **obtain** which is used for explicit ∃-elimination and the term *?thesis* that stands for the current goal, i.e. the enclosing **show** (or **have**) statement.

3 An abstract framework for clock synchronization

In certain distributed systems, e.g., real-time process-control systems, the existence of a reliable global time source is critical in ensuring the correct functioning of the systems. This reliable global time source can be implemented using several physical clocks distributed on different nodes in the distributed system. Since physical clocks are by nature constantly drifting away from the

"real time" and different clocks can have different drift rates, in such a scheme, it is important that these clocks are regularly adjusted so that they are closely synchronized within a certain application-specific safe bound. The design and verification of clock synchronization protocols are often complicated by the additional requirement that the protocols should work correctly under certain types of errors, e.g., failure of some clocks, error in communication network or corrupted messages, etc.

There has been a number of fault-tolerant clock synchronization algorithms studied in the literature, e.g., the *Interactive Convergence Algorithm* (ICA) by Lamport and Melliar-Smith [10], the Lundelius-Lynch algorithm [11], etc., each with its own degree of fault tolerance. One important property that must be satisfied by a clock synchronization algorithm is the agreement property, i.e., at any time t, the difference of the clock readings of any two non-faulty processes must be bounded by a constant (which is fixed according to the domain of applications). At the core of these algorithms is the convergence function that calculates the adjustment to a clock of a process, based on the clock readings of all other processes. Schneider [15] gives an abstract characterization of a wide range of clock synchronization algorithms (based on the convergence functions used) and proves the agreement property in this abstract framework. Schneider's proof was later formalized by Shankar [17] in the theorem prover EHDM, where eleven axioms about clocks are explicitly stated. We re-formalize Schneider's proof in Isabelle/HOL, making use of Shankar's formulation of the clock axioms.

We give an overview of some of the clock axioms of Schneider's generalized clock synchronization algorithm. For more complete and detailed explanation we refer the interested readers to [15,17], in particular [17] for the particular formulation of the clock axioms. Our formalization of Schneider's clock axioms are essentially those of Shankar. The structure of the proof for the main theorem (the agreement property) follows closely Shankar's proof. The whole Isar proof script takes around 1400 lines.

The clock axioms that concern the convergence function used in clock synchronization algorithms are the so-called *translation invariance*, *precision enhancement* and *accuracy preservation* properties. Each of these is explained shortly below, where we denote with *cfn* the abstract convergence function. But first we shall need a few preliminary definitions of types and constants used in our formalization.

Clocks and processes We shall model (physical) clocks as functions from reals to reals, and hence when we speak of *clock values*, we refer to the real numbers. Clock values are denoted with the type *Clocktime*. For readability, we introduce an alias type *time* of *Clocktime* to denote the real time. The set of processes (or nodes) of the distributed system being formalized are given the type *process*. We assume this set of processes is finite and its cardinality is denoted with *np*. Each process maintains its

own *physical clock*. This process-and-physical-clocks pair is denoted with PC in our formalization, which is of type $\mathit{process} \Rightarrow (\mathit{time} \Rightarrow \mathit{Clocktime})$. The *logical clocks* of the processes, i.e., physical clocks with adjustments, are denoted by VC, of the same type as PC. The set of non-faulty processes at time t is denoted with the characteristic function *correct* of type $\mathit{process} \Rightarrow \mathit{time} \Rightarrow \mathit{bool}$.

Clock readings Each process can read the values of the clocks of other processes, the details of exactly how this reading is done are implementation dependent, e.g., via message passing, shared memory, etc. In the abstract framework, clock readings are formalized as functions from processes to clock values. Each process maintains its own clock readings. The pair of a process and its clock readings is denoted with the constant θ of type $\mathit{process} \Rightarrow (\mathit{process} \Rightarrow \mathit{Clocktime})$.

Convergence functions The convergence function *cfn* calculates the adjusted clock value of a process, based on the clock reading of the process. It is of type $\mathit{process} \Rightarrow (\mathit{process} \Rightarrow \mathit{Clocktime}) \Rightarrow \mathit{Clocktime}$.

Synchronization rounds The adjustments to physical clocks of processes take place in rounds. The length of each round is fixed to a certain duration of *logical clock time*, and hence in reality different processes can have different perception on the length of the round. The real time at which a process p reaches a synchronization round i is denoted by $te\ p\ i$, where te is a predicate symbol of type $\mathit{process} \Rightarrow \mathit{nat} \Rightarrow \mathit{bool}$.

The following properties concern requirements of the convergence function.

Translation invariance. This axiom is formally stated as follows: for any positive value x and for any function f mapping clocks to clock values,

$$\mathit{cfn}\ p\ (\lambda n.f(n) + x) = \mathit{cfn}\ p\ f + x$$

This axiom says that the adjusted clock value calculated by the convergence function from a clock reading f with all the readings shifted by x is the same as the adjusted clock value calculated from f, compensated with x.

Precision enhancement. Given any subset C of processes whose cardinality is greater or equal to $np - k$, for some *fault-tolerant degree* k, and given any processes p and q in C, and for any clock readings f and g satisfying the following conditions:

(i) for any $l \in C$, $|f\ l - g\ l| \leq x$,

(ii) for any $l, m \in C$, $|f\ l - f\ m| \leq y$,

(iii) and for any $l, m \in C$, $|g\ l - g\ m| \leq y$,

there is a bound $\pi\ x\ y$ such that $|\mathit{cfn}\ p\ f - \mathit{cfn}\ q\ g| \leq \pi\ x\ y$. Here the value x denotes an upper bound on the difference in the clock readings of f and g and y denotes an upper bound on the difference among the clock readings in f (likewise, g) compared with each other. The precision enhancement

```
constdefs
  okClocks :: [process, process, nat] ⇒ bool
  okClocks p q i ≡ ∀ t. 0 ≤ t ∧ t < max (te p i) (te q i)
      ∧ correct p t ∧ correct q t ⟶ |VC p t − VC q t| ≤ δ

theorem agreement:
  assumes ie1: β ≤ rmin
  and ie2: μ ≤ δS
  and ie3: γ1 δS ≤ δS
  and ie4: γ2 δS ≤ δ
  and ie5: γ3 δS ≤ δ
  and ie6: 0 ≤ t
  and cpq: correct p t ∧ correct q t
shows |VC p t − VC q t| ≤ δ
proof−
  from ie6 cpq event-bound have ∃ i :: nat. t < max (te p i) (te q i)
    by simp
  from this obtain i :: nat where t-bound: t < max (te p i) (te q i) ..
  from t-bound ie1 ie2 ie3 ie4 ie5 four-two have okClocks p q i
    by simp
  from ie6 this t-bound cpq show ?thesis
    by (simp add: okClocks-def)
qed
```

Fig. 1. The agreement theorem in Isabelle/HOL.

axiom asserts that if there are such bounds x and y on the clock readings f and g, then the difference between the adjusted clock values calculated by *cfn* for both clock readings are within some bound $\pi\ x\ y$. The exact function π depends on the concrete convergence function used. However, to guarantee the *agreement property*, this function π needs also to satisfy certain constraints (see the agreement property). The value k, in concrete algorithms, corresponds to the maximum number of faulty processes that can be tolerated in each synchronization round, e.g., for Lundelius-Lynch algorithm we have the constraint $3 \times k + 1 \leq np$.

Accuracy preservation. Given any subset C of processes such that its cardinality is greater or equal to $np - k$, for some *fault-tolerant degree* k, and clock readings f such that for any l and m in C, the bound $|f\ l - f\ m| \leq y$ holds, there is a function α such that for any $p, q \in C$, $|cfn\ p\ f - f\ q| \leq \alpha\ y$.

The agreement property. Assuming that all the axioms on convergence function *cfn* and all the other clock axioms in [17] (among others, the bound on the drift rates of the clocks, the maximum number of faulty process, etc.) are satisfied, for any non-faulty processes p and q and real time t there is a bound δ such that

$$|VC\ p\ t - VC\ q\ t| \leq \delta.$$

Implicit in the above statement is that any real time t always falls into some

synchronization round, a property which needs to be established when proving the agreement theorem. There are various rather technical constraints on the bound δ with respect to other constants, such as the function π in the precision enhancement axiom. We shall not go into details on these constraints and we refer the interested readers to [17,21]. We show an excerpt from the Isar proof script for the agreement property in Figure 1, to illustrate the high-level proof language of Isabelle/Isar.

4 Instances of Schneider's generalized scheme

In this section we give an overview of the proof that the convergence functions of two particular algorithms satisfy the translation invariance, precision enhancement and the accuracy preservation properties discussed in the previous section. The first convergence function that we consider is the *egocentric mean function* of Lamport's Interactive Convergence Algorithm and the second one is the *fault-tolerant midpoint function* of Lundelius-Lynch's algorithm. Both instances have already been verified in the EHDM or PVS system [17,12,16]. Our main objective is to experiment on the combination of Isabelle with other tools by identifying the parts of the proofs that can be done fully automatically by first-order arithmetic provers.

4.1 Interactive Convergence Algorithm (ICA)

The Interactive Convergence Algorithm of Lamport and Melliar-Smith [10] uses a convergence function called the *egocentric mean function*. We show the main parts of the formalization in Isabelle which closely follows [17]. In this algorithm, to calculate the adjustment to the clock of a process p, it compares the value of p's clock with others' and uses this difference to calculate the adjustment. To take into account faulty processes, which might produce large differences in clock readings, a threshold is set up to trim off clock readings which differ too greatly from the clock reading of p. We denote this threshold constant with Δ. We first introduce a constant PR to denote the set of processes, whose cardinality is np.

constdefs
$PR :: process\ set$
$PR \equiv \ \{..np(\}$

The notation $\{..np(\}$ denotes the set of naturals from 0 to $np - 1$. The convergence function is defined using Isabelle's builtin generalized summation over sets.

constdefs
$cfn :: [process, (process \Rightarrow Clocktime)] \Rightarrow Clocktime$

$cfn\ p\ f \equiv (\sum\ l{\in}PR.\ fiX\ f\ p\ l)\ /\ (real\ np)$

The overloaded *real* function is used here to "typecast" the natural *np*. The auxiliary function *fiX* returns the process' own clock reading if the reading of the other processes are more than Δ away.

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

The *translation invariance property* follows from the following lemma, which is proved by Isabelle's induction tactic on np′.

lemma *trans-inv′*: $(\sum\ l{\in}\{..np'(\}.\ fiX\ (\lambda\ y.\ f\ y + x)\ p\ l) =$
$(\sum\ l{\in}\{..np'(\}.\ fiX\ f\ p\ l) + real\ np' * x$

To prove the precision enhancement property, we need to give explicitly the function π (see the previous section for notations) which satisfies the bound set in the precision enhancement axiom. Let c be the cardinality of the set C in the precision enhancement axiom. The bound function $\pi\ x\ y$ in this case is given by

$$\frac{c \times (x + \text{if } y \leq \Delta \text{ then } 0 \text{ else } y) + k \times (2 \times \Delta + x + y)}{np}$$

Note that if $y \leq \Delta$, we obtain the same bound as in [17]. To prove the precision enhancement property we use eight auxiliary lemmas, three of them are about properties on sumation over sets and five lemmas concerning inequalities about the different terms and bounds involved.

For the bounding function $\alpha(y)$ in the accuracy preservation axiom for the ICA algorithm, we use the same function used in [17]:

$$\alpha(y) = y + \frac{k \times \Delta}{np}$$

All the three properties concerning the convergence function are proved in Isabelle (the complete proof script can be found in [1]) and some of these lemmas have also been checked using ICS and CVC-lite. The latter is discussed in Section 5.

4.2 The Lundelius-Lynch algorithm

The convergence function used in the Lundelius-Lynch algorithm takes the midpoint of a *reduced set* of clock readings. More specifically, given the multi-set of clock readings S, the algorithm first removes the k lowest and k highest values, and take the midpoint of the remaining values. Its formalization in

axioms
constants-ax: $1 < np \wedge 2 * khl < np$

constdefs
$PR :: process\ set$
$PR \equiv \{..np(\}$
declare *PR-def*[*simp*]

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

$reduce :: (process \Rightarrow Clocktime) \Rightarrow process\ set \Rightarrow Clocktime\ set$
$reduce\ f\ P \equiv f\ `\ (P - (kmax\ f\ P \cup kmin\ f\ P))$

$cfnl :: process \Rightarrow (process \Rightarrow Clocktime) \Rightarrow Clocktime$
$cfnl\ p\ f \equiv (Max\ (reduce\ f\ PR) + Min\ (reduce\ f\ PR))\ /\ 2$

Fig. 2. The convergence function of Lundelius-Lynch algorithm.

Isabelle/HOL is more complicated than that of ICA, since we need to formalize an abstract notion of k-highest (lowest) values from given clock readings (which, we recall, is formalized as a function from process to clock values). The definition of Lundelius-Lynch's convergence function in Isabelle/HOL is given in Figure 2. Note that the constant *khl* in the figure corresponds to the constant k in the axioms on convergence functions in the previous section, that is, the maximum number of faulty processes tolerated. To formalize the notion of k-lowest (highest) values, we use Hilbert's choice operator (denoted in Isabelle/HOL with the keyword *SOME*). The function *card* computes the cardinality of a set. The notation $f`S$ denotes the set resulting from applying the function f to every element of the set S. This is used in the definition of the reduced sets (the function *reduce* in the figure). The function Max and Min take the maximum and the minimum of a set, respectively. The convergence function is given by the constant *cfnl*. Note that we use Hilbert's choice operator so that we can abstract from any particular data structures used in storing clock readings, e.g., lists or multisets, which are used in the original proof by Lundelius-Lynch [11].

To prove the precision enhancement and the accuracy preservation, we use the bounding functions used to prove the same properties in [12]. That is, for the

precision enhancement, we use the function

$$\pi\ x\ y = \frac{y}{2} + x$$

and for the accuracy preservation, the identity function $\alpha\ y = y$. The complete proof script for the properties of Lundelius-Lynch convergence function can be found in [1].

5 Using external automatic provers as oracles

The proofs for the properties concerning the convergence functions used in ICA and Lundelius-Lynch algorithms require both higher-order reasoning and arithmetics. A large part of the proofs involves linear integer and rational inequalities and equalities. We identify the lemmas that Isabelle's built-in automatic tactics were not able to prove completely and study whether they can be automatically proven by more specific arithmetic solvers. So far, we experimented with ICS and CVC Lite.

ICS (Integrated Canonizer and Solver) [8] is an efficient decision procedure able to decide the satisfiability of formulas in a quantifier-free, first-order theory with arithmetic. It is mainly based on a congruence closure procedure for the theory of equality and disequality with both uninterpreted and interpreted function symbols. It also incorporates a state-of-the-art SAT solver. CVC Lite [2] in addition provides some support for quantifiers.

We illustrate our experience by showing some examples extracted from the conformance proofs of Lynch's fault-tolerant mid point convergence function. So far, the translation from Isabelle/HOL syntax into the specification languages of the external provers is done automatically for quantifier free arithmetic formulas. As shown by the examples below, to eliminate quantifiers we need to do some logical transformations such as quantifier instantiation and skolemization.

We have implemented the translation of quantifier free arithmetic formulas inside Isabelle as an "oracle". The oracle architecture in Isabelle works as follows. The result of the translation is passed to the external prover which is then called for checking. If the external prover declares the formula to be valid, the oracle validates the target formula as an Isabelle theorem. That is, there is *no proof reconstruction* on the Isabelle side.

A translation script exists for both ICS and CVC Lite. Moreover, we translate quantifier free arithmetic formulas in Isabelle/HOL to the SMT-LIB format [14]. The SMT-LIB format is likely to be the future common format for the provers on satisfiability modulo theories (SMT). It is also the format for the challenges of the new SMT competition [3]. This format will certainly become for the SMT solvers what the TPTP [19] and DIMACS [5] formats are for first

[3] See http://combination.cs.uiowa.edu/smtlib/.

```
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 ].
```

Fig. 3. An encoding in ICS.

order provers and SAT solvers, respectively. We have tested this interface with several SMT provers like CVC-Lite, Yices (successor of ICS), Simplics, ARIO and SAMMY among others. All these participated in the SMT competition in 2005 [4].

Syntax translation from Isabelle into the ICS or the SMT-LIB format is mostly trivial for standard arithmetic operators, except the absolute value operator, which needs to be encoded via if-then-else operator. There are, of course, some technical details that have to be considered. For example, the translation incorporates a module for generating new symbols (variables, functions, predicates, sorts) which are supported by the SMT-LIB format. This is needed because naming of variables or functions in Isabelle might not be acceptable in SMT-LIB format (e.g., the use of quotes in polymorphic types) and thus cannot be exported directly to SMT-LIB.

The translations we have done so far can be divided into three categories: pure quantifier free arithmetic formulas, arithmetic formulas involving simple set-theoretic constructs and arithmetic formulas with quantifiers. Each of these categories is illustrated by the following examples.

We start with a simple lemma about the distributivity of absolute value at which all Isabelle automatic tactics fail. The translation for ICS is shown in Figure 3.

lemma *abs-distrib-div*: $0 < (c{::}real) \implies |a\ /\ c - b\ /\ c| = |a - b|\ /\ c$

The ICS language does not include the division operator, so we introduce d as being the inverse of c. The absolute values on each side of the equation are calculated by `abslhs` and `absrhs`, respectively. ICS is able to check the validity of the resulting formula (via refutation of its negation) automatically.

Let us see now a more interesting example. For the proof of the *precision enhancement* property, we need to prove the following lemma:

[4] See seehttp://www.csl.sri.com/users/demoura/smt-comp/.

```
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;
....

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

Fig. 4. An encoding in CVC Lite.

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

The proof in Isabelle is quite long and is done by case analysis on the sign of the term inside the absolute value. Figure 4 shows some parts of our first attempt to encode the above lemma into CVC Lite (the omitted part will be explained in due course).

Notice that since CVC Lite does not support set theoretic reasoning, we need to encode the set-related assumptions using uninterpreted types and constants. Here, `SETPROC` and `PROC` are new types for processes and set of processes. The cardinality of sets is encoded as the uninterpreted function symbol `card`, sets inclusion is encoded as `INCL` and set membership as `INSET`.

The omitted part in Figure 4 contains assumptions needed to establish the main theorem (the statement inside the `QUERY`). We show a few assumptions and how they are dealt with in order to have CVC Lite automatically prove the theorem. The following assumptions concern the clock readings `f` and `g`:

```
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);

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;
```

We launch CVC Lite on this first translation attempt but unfortunately, it fails due to the quantifiers in the assumptions. In the next attempt we instantiate the universal quantifier on `C` in the first two assumptions. In our theory (see Figure 4) there are two such constants `C` and `PR` of the right type. We consider only the instantiation with `C`. The modified assumption `uboundmaxf` (similarly for `uboundmaxg`) becomes

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

However, CVC Lite does not terminate (after two hours execution) on this second version. We then proceed to skolemize the existential quantifiers by introducing new constants, resulting in the following quantifier free assumptions

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

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

Again, the prover does not seem to terminate. We finally eliminate all the universal quantifiers in the remaining assumptions (`hbx, hby1` and `hby2`) by enumerating all possible instantiations:

```
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;
```

Under these modified assumptions, CVC Lite successfully proved the main theorem in Figure 4.

An encouraging example is the *accuracy preservation* property which is successfully proven with both CVC Lite and ICS. In both solvers the proofs finish in less than one second.

6 Future Work

The present paper mainly reports on a case study to investigate the integration of tools to obtain a platform suitable for formal verification of distributed algorithms. Several other case studies have also been done, including the distributed reference counting problem [13] and the Disk Paxos algorithm [7].

In the experience reported here we have followed Shankar's formalization of Schneider's protocol. However some of the clock conditions assumed in this formalization are too strong, and there has been works in refining this formalization while maintaining its generality [12,16]. We plan to study these works as well.

This work is done within the QSL Platforme project at the LORIA laboratory (see http://qsl.loria.fr/). At present, we are investigating how to automatically extract and translate proof objects produced by external tools to Isabelle/Isar proofs. Proof reconstruction would preserve the trust in the theorem prover as opposed to a proof where some parts are done by accepting the verdict of an external prover as oracle. Even if it may be acceptable to trust the external tool, errors might possibly be introduced in the functions for syntax translation.

As future application of the platform we are particularly interested in the FlexRay protocol, where the clock synchronization algorithm used is a variant of Lundelius-Lynch's fault-tolerant midpoint algorithm. Also the start-up phase in this protocol is a challenging verification task and its correctness has, to our knowledge, not yet been studied using formal verification techniques.

Acknowledgment. This work is part of the QSL Platforme project at LORIA. We are grateful for the comments and suggestions from members of the project: Stephan Merz, Pascal Fontaine, Jean-Yves Marion, Antonia Balaa and Mauro-Javier Jaskelioff.

References

[1] D. Barsotti. Instances of schneider's generalized protocol of clock synchronization. Available on the web, 2005.

[2] CVC Lite. `http://chicory.stanford.edu/CVC/`.

[3] L. de Moura. SAL: Tutorial. Computer Science Laboratory, SRI International, April 2004.

[4] L. A. Dennis, G. Collins, R. Boulton, K. Slind, G. Robinson, M. Gordon, and T. Melham. The PROSPER toolkit. In S. Graf and M. Schwartzbach, editors, *Proceedings of TACAS'03*, number 1785 in LNCS, pages 78 – 92. Springer, June 2003.

[5] DIMACS. *Satisfiability suggested format.*

[6] FlexRay Consortium. *FlexRay Communications System Protocol Specification Version 2.0*, June 2004.

[7] E. Gafni and L. Lamport. Disk paxos. In *DISC '00: Proceedings of the 14th International Conference on Distributed Computing*, pages 330–344. Springer-Verlag, 2000.

[8] Integrated canonizer and solver (ICS). `http://www.icansolve.com/`.

[9] Isabelle home page. `http://isabelle.in.tum.de/`.

[10] L. Lamport and P. M. Melliar-Smith. Synchronizing clocks in the presence of faults. *J. ACM*, 32(1):52–78, 1985.

[11] J. Lundelius and N. Lynch. A new fault-tolerant algorithm for clock synchronization. In *Proceedings of PODC '84*, pages 75–88, New York, NY, USA, 1984. ACM Press.

[12] P. S. Miner. Verification of fault-tolerant clock synchronization systems. NASA Technical Paper 3349, NASA Langley Research Center, November 1993.

[13] L. Moreau and J. Duprat. A construction of distributed reference counting. *Acta Inf.*, 37(8):563–595, 2001.

[14] S. Ranise and C. Tinelli. The SMT-LIB standard : Version 1.1, Mar. 2005.

[15] F. B. Schneider. Understanding protocols for Byzantine clock synchronization. Technical Report TR 87–859, Cornell University, 1987.

[16] D. Schwier and F. von Henke. Mechanical verification of clock synchronization algorithms. In A. P. Ravn and H. Rischel, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, number 1486 in LNCS, pages 262–271. Springer, September 1998.

[17] 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.

[18] J. H. Siekmann, C. Benzmüller, V. Brezhnev, L. Cheikhrouhou, A. Fiedler, A. Franke, H. Horacek, M. Kohlhase, A. Meier, E. Melis, M. Moschner, I. Normann, M. Pollet, V. Sorge, C. Ullrich, C.-P. Wirth, and J. Zimmer. Proof development with omega. In *CADE*, pages 144–149, 2002.

[19] G. Sutcliffe and C. Suttner. The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.

[20] B. Tavernier. Calife: A generic graphical user interface for automata tools. *Electr. Notes Theor. Comput. Sci.*, 110:169–172, 2004.

[21] A. Tiu. A formalization of a generalized clock synchronization protocol in Isabelle/HOL. Available on the web at http://www.loria.fr/~tiu, 2005.