

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

Damian Barsotti¹, Leonor Prensa Nieto² and Alwen Tiu³

¹Universidad Nacional de Córdoba, Ciudad Universitaria, 5000 Córdoba, Argentina

²LORIA, 54506 Vandoeuvre-lès-Nancy, France

³Research School of Information Sciences and Engineering

Australian National University and National ICT Australia, Canberra, ACT 0200, Australia

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 [Sch87] 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 [LMS85] and the Fault-tolerant Midpoint algorithm of Lundelius-Lynch [LL84], satisfy Schneider’s general conditions for correctness. The proofs are completely formalized in Isabelle/HOL. We identify 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 arithmetics.

Keywords: Theorem proving, verification, clock synchronization, combination of deductive tools.

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 [Isa] to include sophisticated decision procedures needed for various verification tasks. On the other hand, there exist many specialized efficient automated provers that handle specific logics, e.g., model checkers, SAT solvers, etc. It is

Correspondence and offprint requests to: Leonor Prensa Nieto and Alwen Tiu

then natural 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 proofs involved. 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 Yices [Yic] and CVC Lite [CVC] 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 Yices 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 [SBB⁺02], the CALIFE project [Tav04], the SAL framework [dM04] and the PROSPER project [DCB⁺03]. 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 studying combination of deductive tools, a substantial part of the work is the formalization of the generalized protocol of clock synchronization by Schneider [Sch87] and the formalization of the correctness of the convergence functions used in the ICA [LMS85] and the Lundelius-Lynch [LL84] clock synchronization algorithms. Verification of these algorithms has been done previously, notably by Shankar [Sha92] in the EHDm theorem prover and by Miner [Min93] and Schwier and von Henke [SvH98] in the theorem prover PVS. But to our knowledge, ours is the first verification of these algorithms from which complete formal proofs, in the form of a natural deduction proof, can be extracted and checked 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 [Fle04] (for drive-by-wire application in automotive industries). There has indeed been on-going work in the verification of FlexRay using Isabelle/HOL [BKKS05, KS06] 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 discusses an implementation of an interface within the Isabelle/HOL system to allow for invoking external SMT solvers to help proving arithmetic statements in Isabelle/HOL. Some motivating examples on the use of the SMT interface are also given, which are taken from the “informal proof” (as opposed to machine verified proofs) of clock synchronization for the Lundelius-Lynch algorithm [LL84]. Section 4 gives an overview of the clock synchronization problem and the formalization of Schneider’s generalized protocol for clock synchronization and presents the first case study of our effort in using a combination of Isabelle/HOL with external SMT tools. Section 5 presents the other two case studies: the formalization of the ICA protocol and the Lundelius-Lynch protocol. These two algorithms are viewed as special instances of the generalized algorithm of Schneider. In each case we show that external tools such as CVC-Lite and Yices can be used to help verifying some lemmas in the verification. Section 6 concludes the paper and discusses future work. The implementation and the case studies presented in this paper are available online.¹

¹ Via <http://gforge.inria.fr/projects/isabelle-smtlib/>

2. Isabelle/HOL and Isar

Isabelle [Isa] is a generic interactive proof assistant. We use Isabelle/HOL which is the instance for higher-order logic. Isabelle is widely 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. The main drawback of such interactive 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* try a combination of the precedent ones.

Isabelle’s meta-logic comes with a type of *propositions* with implication \implies and a universal quantifier \bigwedge for expressing inference rules and generality. 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 ‘ \equiv ’.

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. 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 $\textit{formula-0} \implies \textit{formula-n+1}$. A statement is announced as a **theorem** or **lemma** (free choice with no formal difference between them) where (possibly labeled) assumptions follow the keyword **assumes** and the conclusion is preceded by **shows**. Proofs are enclosed between **proof** and **qed**. Within a proof **assume** augments the proof context with a (labeled) assumption, the **haves** in between state intermediate results that help proving the current goal and **show** solves the current goal (in the sketch above it is the conclusion of the theorem). The keyword **proof** announces the beginning of a 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-** instead. The keyword **by** *m* abbreviates **proof** *m* **qed** and is used for “one-line” proofs. The tactics *simp* and *blast* are typical built-in proof engines of Isabelle. The “..” stands for a proof done basically by applying a rule from a predefined set of introduction rules. When the proof of a statement follows from a number of facts, these can be fed into a proof using **from**. The term *?thesis* stands for the current goal, i.e. that of the enclosing **show** (or **have**) statement.

```

lemma example: formula-0
proof
  have l1: formula-1 by simp
  have l2: formula-2 by simp
  ...
  have ln: formula-n by simp
  from l1 ... ln show ?thesis by simp
qed

```

Having to introduce names for all the elements needed in such a sequence can be avoided by separating their proofs with **moreover** and using **ultimately** to reconsider the so accumulated facts. Already existing facts can be recalled and added to such sequences using **note**.

```

lemma example: formula-0

```

```

proof
  have formula-1 by simp
  moreover
  have formula-2 by simp
  ...
  moreover
  have formula-n by simp
  moreover
  note existing-facts
  ultimately show ?thesis by simp
qed

```

If at each step we only need to recall the immediately preceding proposition, we can refer to it via the fact binding *this*. Because of the frequency of **from** *this*, Isar offers some elegant abbreviations:

```

then = from this
thus = then show
hence = then have

```

Introducing new local variables (parameters) into a proof is done with the command **fix**. Such fixed variables correspond to meta-quantified variables. An Isabelle statement $\bigwedge x_1, \dots, x_n. \llbracket A_1; \dots; A_n \rrbracket \implies A$ with parameters x_1, \dots, x_n , assumptions A_1, \dots, A_n and conclusion A can be written in Isar as

```

fix  $x_1 \dots$  fix  $x_m$ 
assume  $A_1 \dots$  assume  $A_n$ 
show  $A$ 

```

To introduce and prove such a statement within a proof we can simply enclose it together with its proof within a *raw proof block* which is delimited by braces “{” and “}”.

```

{fix  $x_1 \dots$  fix  $x_m$ 
  assume  $A_1 \dots$  assume  $A_n$ 
  have  $A$  proof }

```

The conclusion in such blocks is stated with **have** because it does not correspond to some pending goal but to a fresh introduced statement.

Finally, existence reasoning by explicit \exists -elimination is done via the command **obtain**

```

from  $\exists x. P x$  obtain  $a$  where  $P a ..$ 

```

This text follows the mathematical style of concluding $P a$ from $\exists x. P x$, while carefully introducing a as a new local variable.

These are basically the Isar features encountered in this paper. For a structured presentation of the language, we refer the interested reader to [Nip02].

3. An SMT oracle in Isabelle/HOL

The core of the Isabelle theorem prover is a natural deduction proof system, which is, roughly speaking, a set of inference rule schemes. Isabelle’s kernel provides functions that manipulate formulas and terms using a set of primitives that correspond to the natural deduction system underlying the prover. As a consequence, the consistency of the whole prover depends only on the kernel that implements the natural deduction system. Since the kernel is quite a small set of codes, human readers can understand the working of the prover by inspecting the codes. This increases the confidence in the correct functioning of the system. However, reducing proofs of every theorem to the small set of inference rules also means that interfacing with external tools might be difficult, since every proof produced by the external tools must be converted to a proof based on Isabelle’s proof system. To circumvent this problem, Isabelle allows a ‘quick-and-dirty’ mode which bypass the kernel. This is done via an *oracle* interface: one can program any function for proving a theorem and uses the interface to register the function as a proof method of Isabelle. We use the oracle facility to build an interface to external provers. Since the theorems constructed via oracles bypass Isabelle’s kernel, the trusted

code base is no longer confined to the kernel but also include the source codes of the external tools. The ideal solution would be to translate proofs produced by the external tools; we will leave that to future work.

In the current work we use the oracle facility of Isabelle to interface Isabelle with external SMT (Satisfiability Modulo Theories) automated provers. The logics behind SMT provers are extensions of first order logic with theories, such as equality, (linear) arithmetic, theories for lists, arrays, sets, etc. We focus on tools that support first order logic with real and integer arithmetics since this logic is supported by most of the SMT provers currently available and a significant part of our verification problems fall into this domain. For our experiment, we use the SMT provers that participated in the SMT-COMP 2005 competition², in particular CVC-Lite, Yices and Ario³.

The interface between Isabelle and the above mentioned SMT provers is done by translating Isabelle’s theorems into the SMTLIB format [RT05], a common format of input for SMT provers. We are following version 1.1 of SMTLIB used in the SMT-COMP 2005. We do not implement the whole SMTLIB format; rather we focus on the fragment of quantifier-free first order formulas with equality and arithmetics. In the SMTLIB syntax one can define new type symbols, variables, function and predicate symbols of first-order types (i.e., functions or predicates which take other functions as arguments are not allowed). Expressions in SMTLIB are written in postfix notation. For the details of the format, interested readers are referred to the paper [RT05]. The translation from Isabelle’s syntax to SMTLIB is mostly straightforward. One exceptional case is the translation of the absolute value operator. The expression $|t|$, where t is an arithmetic expression, is translated to SMTLIB via the if-then-else construct, that is, the expression:

$$\text{if } t \geq 0 \text{ then } t \text{ else } -t.$$

In the SMTLIB format, one would write

```
(ite (>= t 0) t (- t))
```

where `ite` denotes the “if-then-else” operator. Additionally, we need to add an assumption stating that the absolute value is greater than or equal to zero:

```
(>= (ite (>= t 0) t (- t)) 0)
```

Another exceptional case is set related operators, e.g., set membership, and set inclusion. In the current work we translate these operators to uninterpreted predicate symbols.

The implementation of the interface to SMT solvers is done in Isabelle as a tactic (i.e., a function written in the ML language) which is registered as an oracle called `smt`. The interface translates Isabelle’s theorems to SMTLIB, save the translation to a file, and then call the external SMT solver to solve it. The result returned by the solver is a simple yes/no answer, whose form may differ from one prover to another, e.g., CVC-Lite returns “Valid” for a valid theorem while Yices returns “unsat”, etc. We now show several motivating examples on the use of the SMT oracle to illustrate the benefit of using the external tools.

Example: simple arithmetics For a starter, let us consider a statement in arithmetics which are “obviously true” for anyone familiar with high-school arithmetics, that is, the formula

$$x * (1 + y) - x * (1 - y) = 2 * x * y.$$

Notice that the equation is non-linear, and its proof is out of the scope of the available built-in tactics in Isabelle which can handle only linear arithmetic. The “manual proof” (as opposed to automated proofs) of this equation, in the Isar proof language, is given in Figure 1. In the figure, the keyword `simp` refers to the built-in simplifier in Isabelle. The keywords `real-mult-commute` and others within the `by` command refer to lemmas provided in the HOL library in Isabelle. The above equation is proved immediately using the SMT oracle with the external tool CVC-Lite:

```
lemma obvious: (x :: real)*(1 + y) - x*(1 - y) = 2*x*y by smt
```

CVC-Lite is the only prover, among the SMT provers we tested, that is capable of proving this formula. Other provers do not handle non-linear arithmetics. Similar equations appear throughout the correctness

² See <http://combination.cs.uiowa.edu/smtlib/> for a complete list of participants of SMT-COMP 2005.

³ The choice of these tools is motivated by their relatively good performance in the SMT-COMP and their robustness in handling the SMTLIB format.

```

lemma obvious: (x :: real)*(1 + y) - x*(1 - y) = 2*x*y
proof-
  have x*(1 + y) = x + x*y
  proof-
    have x*(1 + y) = (1 + y)*x
      by(simp add: real-mult-commute)
    moreover
    have (1 + y)*x = x + y*x
      by (simp add: real-add-mult-distrib)
    moreover
    have x + y*x = x + x*y by simp
    ultimately show ?thesis by simp
  qed
  moreover
  have x*(1 - y) = x - x*y
  proof-
    have x*(1 - y) = (1 - y)*x by simp
    moreover
    have (1 - y)*x = (1 + -y)*x by simp
    moreover
    have (1 + -y)*x = x - y*x by (simp add: real-add-mult-distrib)
    ultimately show ?thesis by simp
  qed
  ultimately show ?thesis by simp
qed

```

Fig. 1. An Isar proof script of a simple arithmetic formula.

proof of Lundelius-Lynch algorithm presented in [LL84]. For a more complicated example, consider the equation (which appears in [LL84], page 15):

$$U^i + (1 + \rho) * (\beta + \epsilon) + \rho * \delta = U^i + (2 * (1 + \rho) * (\beta + \epsilon) + (1 + \rho) * \delta + \delta * \rho) - (1 + \rho) * (\beta + \delta + \epsilon).$$

Its formal proof in Isabelle requires a number of applications of the distributivity of multiplication over summation with appropriate abstractions of the summands at each step of the applications. On the contrary, the SMT oracle (with CVC-Lite as the background engine) proves this equation immediately.

Example: handling quantifiers and definitions. In this example we show how we use the SMT oracle to prove lemmas involving first-order structures and arithmetics. Again, this example is taken from the analysis on Lundelius-Lynch algorithm. One of the basic assumptions in this algorithm is that concerning the behaviours of physical clocks. The precision of physical clocks is assumed to be bounded within a constant factor ρ , which is called the *drift rate*. Time is formalized as real numbers and physical clocks are formalized as functions from real time to clock time, which are basically functions from reals to reals. Let C denote a physical clock, then C satisfies the following requirements: if $t_1 \leq t_2$ then

$$(t_2 - t_1)/(1 + \rho) \leq C t_2 - C t_1 \leq (1 + \rho) * (t_2 - t_1),$$

$$(1 - \rho) * (t_2 - t_1) \leq C t_2 - C t_1 \leq (t_2 - t_1)/(1 - \rho)$$

and $C t_1 \leq C t_2$. The latter is the monotonicity property. These assumptions are formalized in Isabelle via the following definitions:

```

rho-bounded1 C ≡ ∀ s t. s ≤ t → (C t - C s) ≤ (t - s) * (1 + rho)
rho-bounded2 C ≡ ∀ s t. s ≤ t → (t - s) ≤ (C t - C s) * (1 + rho)
rho-bounded3 C ≡ ∀ s t. s ≤ t → (t - s) * (1 - rho) ≤ (C t - C s)
rho-bounded4 C ≡ ∀ s t. s ≤ t → (C t - C s) * (1 - rho) ≤ (t - s)
rho-bounded C ≡ rho-bounded1 C ∧ rho-bounded2 C ∧ rho-bounded3 C ∧ rho-bounded4 C
monotone C ≡ ∀ t1 t2. t1 ≤ t2 → C t1 ≤ C t2

```

Now let us consider the following property of physical clocks (Lemma 2(a) in [LL84]):

$$|(C t_2 - t_2) - (C t_1 - t_1)| \leq \rho * |t_2 - t_1|.$$

It is enough to consider the case where $t_1 \leq t_2$, since the other case is symmetric. This property is formalized in Isabelle as the following lemma:

```

lemma LL2a:
assumes ti:  $t1 \leq t2$ 
and rb: rho-bounded C
and mnt: monotone C
shows  $|(C\ t2 - t2) - (C\ t1 - t1)| \leq \text{rho} * |t2 - t1|$ 
proof-
  note ti
  moreover
  from ti mnt have  $C\ t1 \leq C\ t2$  by (simp add: monotone-def)
  moreover
  from ti rb have  $(C\ t2 - C\ t1) \leq (t2 - t1) * (1 + \text{rho})$ 
    by (simp add: rho-bounded-def rho-bounded1-def)
  moreover
  from ti rb have  $(t2 - t1) \leq (C\ t2 - C\ t1) * (1 + \text{rho})$ 
    by (simp add: rho-bounded-def rho-bounded2-def)
  moreover
  from ti rb have  $(t2 - t1) * (1 - \text{rho}) \leq (C\ t2 - C\ t1)$ 
    by (simp add: rho-bounded-def rho-bounded3-def)
  moreover
  from ti rb have  $(C\ t2 - C\ t1) * (1 - \text{rho}) \leq (t2 - t1)$ 
    by (simp add: rho-bounded-def rho-bounded4-def)
  ultimately show ?thesis by smt
qed

```

Fig. 2. Unfolding definitions and instantiations of quantifiers.

```

lemma LL2a:
assumes ti:  $t1 \leq t2$ 
and rb: rho-bounded C
and mnt: monotone C
shows  $|(C\ t2 - t2) - (C\ t1 - t1)| \leq \text{rho} * |t2 - t1|$ 

```

To prove this lemma, one must first unfold the definitions ‘rho_bounded’ and ‘monotone’ and instantiate the universal quantifiers in the definitions. In this case, the instantiations are quite easy to guess. Once the unfolding and the instantiation have been done, we prove the resulting formula using the SMT oracle. The Isar proof script for this lemma is shown in Figure 2. The proof is considerably shorter than the proof without the SMT oracle, which uses four other lemmas and took approximately one hour to write down. The use of the SMT oracle instead requires only a few minutes to write down and less than a second runtime. Of course in this case the instantiations of the quantifiers are rather easy. In more complicated cases some ingenuity might be required to come up with the appropriate instantiations. In these cases, the use of SMT oracle is useful only in the case where the resulting quantifier-free arithmetic formulas are complex and require lengthy manual proofs. In Section 5 we shall encounter some of these cases.

4. 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 [LMS85], the Lundelius-Lynch

algorithm [LL84], 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 [Sch87] 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 [Sha92] in the theorem prover EHDM, where eleven axioms about clocks are explicitly stated. We 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 a more complete and detailed explanation we refer the interested reader to [Sch87, Sha92], in particular [Sha92] for the 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, but the detailed proof of each lemma is done independently. This is mainly because the original formulation of Shankar's proof was done in a completely different system and was not accessible to the authors. The whole Isar proof script takes around 1400 lines and is available from Isabelle's proof archive [Tiu05].

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 $process \Rightarrow (time \Rightarrow Clocktime)$. 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 $process \Rightarrow time \Rightarrow bool$.

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 $process \Rightarrow (process \Rightarrow Clocktime)$.

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 $process \Rightarrow (process \Rightarrow Clocktime) \Rightarrow Clocktime$.

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 $process \Rightarrow nat \Rightarrow bool$.

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,

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

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:

types

process = *nat*
event = *nat*
time = *real*
Clocktime = *real*

consts

$\delta :: \text{real}$
 $\mu :: \text{real}$
 $\varrho :: \text{real}$
 $rmin :: \text{real}$
 $rmax :: \text{real}$
 $\beta :: \text{real}$
 $\Lambda :: \text{real}$

np :: *process*
maxfaults :: *process*

PC :: [*process*, *time*] \Rightarrow *Clocktime*
VC :: [*process*, *time*] \Rightarrow *Clocktime*
te :: [*process*, *event*] \Rightarrow *time*
 ϑ :: [*process*, *event*] \Rightarrow (*process* \Rightarrow *Clocktime*)
IC :: [*process*, *event*, *time*] \Rightarrow *Clocktime*
correct :: [*process*, *time*] \Rightarrow *bool*
cfn :: [*process*, (*process* \Rightarrow *Clocktime*)] \Rightarrow *Clocktime*
 π :: [*Clocktime*, *Clocktime*] \Rightarrow *Clocktime*
 α :: *Clocktime* \Rightarrow *Clocktime*

constdefs

count :: [*process* \Rightarrow *bool*, *process*] \Rightarrow *nat*
count f n \equiv *card* {*p*. *p* < *n* \wedge *f p*}

Adj :: [*process*, *event*] \Rightarrow *Clocktime*
Adj \equiv (λ *p i*. *if* 0 < *i* *then* *cfn p* (ϑ *p i*) - *PC p* (*te p i*)
 else 0)

okRead1 :: [*process* \Rightarrow *Clocktime*, *Clocktime*, *process* \Rightarrow *bool*] \Rightarrow *bool*
okRead1 f x ppred \equiv \forall *l m*. *ppred l* \wedge *ppred m* \longrightarrow $|f\ l - f\ m| \leq x$

okRead2 :: [*process* \Rightarrow *Clocktime*, *process* \Rightarrow *Clocktime*, *Clocktime*,
 process \Rightarrow *bool*] \Rightarrow *bool*
okRead2 f g x ppred \equiv \forall *p*. *ppred p* \longrightarrow $|f\ p - g\ p| \leq x$

rho-bound1 :: [[*process*, *time*] \Rightarrow *Clocktime*] \Rightarrow *bool*
rho-bound1 C \equiv \forall *p s t*. *correct p t* \wedge *s* \leq *t* \longrightarrow *C p t* - *C p s* \leq (*t* - *s*) * (*1* + ϱ)
rho-bound2 :: [[*process*, *time*] \Rightarrow *Clocktime*] \Rightarrow *bool*
rho-bound2 C \equiv \forall *p s t*. *correct p t* \wedge *s* \leq *t* \longrightarrow (*t* - *s*) * (*1* - ϱ) \leq *C p t* - *C p s*

Fig. 3. Types and constants definition for Schneider's framework.

1. for any $l \in C$, $|f\ l - g\ l| \leq x$,
2. for any $l, m \in C$, $|f\ l - f\ m| \leq y$,
3. and for any $l, m \in C$, $|g\ l - g\ m| \leq y$,

there is a bound $\pi\ x\ y$ such that $|cfn\ p\ f - cfn\ q\ g| \leq \pi\ x\ y$. 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 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

axioms

constants-ax: $0 < \beta \wedge 0 < \mu \wedge 0 < rmin$
 $\wedge rmin \leq rmax \wedge 0 < \rho \wedge 0 < np \wedge maxfaults \leq np$

PC-monotone: $\forall p s t. correct\ p\ t \wedge s \leq t \longrightarrow PC\ p\ s \leq PC\ p\ t$

VClock: $\forall p t i. correct\ p\ t \wedge te\ p\ i \leq t \wedge t < te\ p\ (i + 1) \longrightarrow VC\ p\ t = IC\ p\ i\ t$

IClock: $\forall p t i. correct\ p\ t \longrightarrow IC\ p\ i\ t = PC\ p\ t + Adj\ p\ i$

Condition 2: bounded drift

axioms

rate-1: $\forall p s t. correct\ p\ t \wedge s \leq t \longrightarrow PC\ p\ t - PC\ p\ s \leq (t - s) * (1 + \rho)$

rate-2: $\forall p s t. correct\ p\ t \wedge s \leq t \longrightarrow (t - s) * (1 - \rho) \leq PC\ p\ t - PC\ p\ s$

Condition 9: Translation invariance

axioms

trans-inv: $\forall p f x. 0 \leq x \longrightarrow cf_n\ p\ (\lambda y. f\ y + x) = cf_n\ p\ f + x$

Condition 10: precision enhancement

axioms

prec-enh:

$\forall p p_{pred}\ p\ q\ f\ g\ x\ y.$
 $np - maxfaults \leq count\ p_{pred}\ np \wedge$
 $okRead1\ f\ y\ p_{pred} \wedge okRead1\ g\ y\ p_{pred} \wedge$
 $okRead2\ f\ g\ x\ p_{pred} \wedge p_{pred}\ p \wedge p_{pred}\ q$
 $\longrightarrow abs(cf_n\ p\ f - cf_n\ q\ g) \leq \pi\ x\ y$

Condition 11: accuracy preservation

axioms

acc-prsv:

$\forall p p_{pred}\ p\ q\ f\ x. okRead1\ f\ x\ p_{pred} \wedge np - maxfaults \leq count\ p_{pred}\ np$
 $\wedge p_{pred}\ p \wedge p_{pred}\ q \longrightarrow abs(cf_n\ p\ f - f\ q) \leq \alpha\ x$

Fig. 4. Some clock conditions.

$np - k$, 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 $p, q \in C$, $|cf_n\ p\ f - f\ q| \leq \alpha\ y$.

Figure 3 shows the formalization in Isabelle of various constants and types used in Schneider’s framework. Some of these constants, in particular, the ‘rho-bounded’ constant will be used in subsequent formalization of the Lundelius-Lynch and the ICA protocols. The type ‘event’ represents synchronization rounds. The constants PC encodes physical clocks, IC encodes the of adjusted physical clocks per synchronization round and VC encodes the virtual clocks (i.e., clocks displayed by correct processes). The constant ‘Adj’ represents the adjustment function that calculates the amount of adjustment to the physical clocks, given the convergence function cf_n . The definition of ‘rho-bounded’ is similar the one in Section 3, except that now we take into account the non-faultiness of the clocks. Figure 4 presents some assumptions on the behaviour of clocks mentioned previously. The full formalization can be found in [Tiu05].

The agreement property. Assuming that all the axioms on convergence function cf_n and all the other clock axioms in [Sha92] (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. The formalized statement of the agreement property is given in Figure 5. We shall not explain in details how these constraints are derived. We refer the interested readers to [Sha92].

```

constdefs
   $\gamma 1 :: \text{real} \Rightarrow \text{real}$ 
   $\gamma 1 x \equiv \pi (2 * \rho * \beta + 2 * \Lambda) (2 * \Lambda + x + 2 * \rho * (rmax + \beta))$ 
   $\gamma 2 :: \text{real} \Rightarrow \text{real}$ 
   $\gamma 2 x \equiv x + 2 * \rho * rmax$ 
   $\gamma 3 :: \text{real} \Rightarrow \text{real}$ 
   $\gamma 3 x \equiv \alpha (2 * \Lambda + x + 2 * \rho * (rmax + \beta)) + \Lambda + 2 * \rho * \beta$ 
theorem agreement:
  assumes  $ie1: \beta \leq rmin$ 
  and  $ie2: \mu \leq \delta S$ 
  and  $ie3: \gamma 1 \delta S \leq \delta S$ 
  and  $ie4: \gamma 2 \delta S \leq \delta$ 
  and  $ie5: \gamma 3 \delta S \leq \delta$ 
  and  $ie6: 0 \leq t$ 
  and  $cpq: \text{correct } p \ t \wedge \text{correct } q \ t$ 
shows  $|VC \ p \ t - VC \ q \ t| \leq \delta$ 

```

Fig. 5. The agreement property.

```

lemma bounded-drift:
  assumes  $st: s \leq t$ 
  and  $rb1: \text{rho-bound1 } C$ 
  and  $rb2: \text{rho-bound2 } C$ 
  and  $rb3: \text{rho-bound1 } D$ 
  and  $rb4: \text{rho-bound2 } D$ 
  and  $corr p: \text{correct } p \ t$ 
  and  $corr q: \text{correct } q \ t$ 
shows  $|C \ p \ t - D \ q \ t| \leq |C \ p \ s - D \ q \ s| + 2 * \rho * (t - s)$ 
proof -
  note  $st$ 

  moreover
  from  $st \ corr p \ rb1$  have  $C \ p \ t - C \ p \ s \leq (t - s) * (1 + \rho)$ 
  by ( $\text{auto simp add: rho-bound1-def}$ )

  moreover
  from  $st \ corr p \ rb2$  have  $(t - s) * (1 - \rho) \leq C \ p \ t - C \ p \ s$ 
  by ( $\text{auto simp add: rho-bound2-def}$ )

  moreover
  from  $st \ corr q \ rb1$  have  $C \ q \ t - C \ q \ s \leq (t - s) * (1 + \rho)$ 
  by ( $\text{auto simp add: rho-bound1-def}$ )

  moreover
  from  $st \ corr q \ rb2$  have  $(t - s) * (1 - \rho) \leq C \ q \ t - C \ q \ s$ 
  by ( $\text{auto simp add: rho-bound2-def}$ )

  moreover
  from  $st \ corr p \ rb3$  have  $D \ p \ t - D \ p \ s \leq (t - s) * (1 + \rho)$ 
  by ( $\text{auto simp add: rho-bound1-def}$ )
  moreover
  from  $st \ corr p \ rb4$  have  $(t - s) * (1 - \rho) \leq D \ p \ t - D \ p \ s$ 
  by ( $\text{auto simp add: rho-bound2-def}$ )

  moreover
  from  $st \ corr q \ rb3$  have  $D \ q \ t - D \ q \ s \leq (t - s) * (1 + \rho)$ 
  by ( $\text{auto simp add: rho-bound1-def}$ )
  moreover
  from  $st \ corr q \ rb4$  have  $(t - s) * (1 - \rho) \leq D \ q \ t - D \ q \ s$ 
  by ( $\text{auto simp add: rho-bound2-def}$ )
  ultimately show  $?thesis$ 
  by( $smt$ )
qed

```

Fig. 6. A proof of the bounded-drift lemma using the SMT oracle.

We now show how the SMT oracle is used to help proving some parts of the proof of the key lemmas in this formalization of Schneider's framework. We show here the main interesting applications.

Calculating bounded drifts. One of the key lemma in establishing the agreement property is that for any two non-faulty processes, the difference in their (logical) clock readings at any given time interval is bounded by a constant factor of ρ , provided that the clocks satisfy the 'rho-bounded' property. The precise statement in Isabelle is as follows.

lemma *bounded-drift*:

```

assumes ie:  $s \leq t$ 
and rb1: rho-bound1 C
and rb2: rho-bound2 C
and rb3: rho-bound1 D
and rb4: rho-bound2 D
and corr-p: correct p t
and corr-q: correct q t
shows  $|C\ p\ t - D\ q\ t| \leq |C\ p\ s - D\ q\ s| + 2*\rho*(t - s)$ 

```

We prove this lemma using the SMT oracle with CVC-Lite as the proof engine. As in the example in the previous section, we need to unfold the definitions in the lemma and instantiate the universal quantifications in those definitions manually. The proof of this lemma is given in Figure 6.

Handling case analyses in arithmetics. We wish to show that a certain bound holds for the physical clock readings in any synchronization round i . In the following, the term $te\ p\ i$ denotes the real time when the process p starts the synchronization round i . The statement of the lemma is as follows:

lemma *beta-rho*:

```

assumes ie:  $te\ q\ (i+1) \leq te\ p\ (i+1)$ 
and corr-p: correct p (te p (i+1))
and corr-q: correct q (te p (i+1))
and corr-l: correct l (te p (i+1))
shows  $|(PC\ l\ (te\ p\ (i+1)) - PC\ l\ (te\ q\ (i+1))) - (te\ p\ (i+1) - te\ q\ (i+1))| \leq \beta*\rho$ 

```

The proof of this lemma requires some intermediate lemmas. We first define some abbreviations, using the Isabelle/Isar syntax, for some subterms of the above lemma.

```

let ?X =  $(PC\ l\ (te\ p\ (i+1)) - PC\ l\ (te\ q\ (i+1)))$ 
let ?D =  $te\ p\ (i+1) - te\ q\ (i+1)$ 

```

The intermediate lemmas that are needed are as follows:

```

posD:  $0 \leq ?D$ 
posX:  $0 \leq ?X$ 
bound1:  $?X \leq ?D * (1 + \rho)$ 
bound2:  $?D*(1 - \rho) \leq ?X$ 
Factor2:  $(x::real)*(1 - (y::real)) = x - x*y$ 

```

Given these intermediate lemmas, the preceding lemma can be proved by case analysis on the relation between the term $?D$ and $?X$:

```

show  $|(PC\ l\ (te\ p\ (i+1)) - PC\ l\ (te\ q\ (i+1))) - (te\ p\ (i+1) - te\ q\ (i+1))| \leq \beta*\rho$ 

```

proof *cases*

```

assume A:  $?D \leq ?X$ 
from posX posD A have absEq:  $|\?X - ?D| = ?X - ?D$ 
by(simp add: real-abs-def)
from bound1 have bound2:  $?X - ?D \leq ?D*\rho$ 
by(simp add: real-mult-commute real-add-mult-distrib)
from D-beta absEq bound2 show ?thesis by simp
next
assume notA:  $\neg (?D \leq ?X)$ 
from this have absEq2:  $|\?X - ?D| = ?D - ?X$ 
by(simp add: real-abs-def)
from ie corr-l rate-2 have bound3:  $?D*(1 - \rho) \leq ?X$  by simp
from this have  $?D - ?X \leq ?D*\rho$  by (simp add: Factor2)
from this absEq2 D-beta show ?thesis by simp

```

qed
qed

Using the SMT oracle with CVC-Lite prover, however, this lemma is proved directly as follows:

```
from posD posX bound1 bound2 D-beta
  show |(PC l (te p (i+1)) - PC l (te q (i+1))) - (te p (i+1) - te q (i+1))| ≤ β*ρ by smt
```

Notice that the ‘Factor2’ lemma is not needed in this case.

5. 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 the 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 EHD or PVS system [Sha92, Min93, SvH98]. However, our proofs differ slightly from previously existing proofs, in particular, we use a different bounding function π (see Section 4) for the precision enhancement lemma for the ICA protocol. 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. The complete formal proofs in this section, without the use of the SMT oracle, can be found in [Bar06].

5.1. Interactive Convergence Algorithm (ICA)

The Interactive Convergence Algorithm of Lamport and Melliar-Smith [LMS85] uses a convergent function called the *egocentric mean function*. We show the main parts of the formalization in Isabelle which closely follows [Sha92]. 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 ≡ {..np{}
```

The notation $\{..np\}$ denotes the set of natural numbers from 0 to $np - 1$. The convergence function is defined using Isabelle’s built-in generalized summation over sets.

```
constdefs
  cfni :: [process, (process ⇒ Clocktime)] ⇒ Clocktime
  cfni p f ≡ (∑ l∈{..np{}. fiX f p l) / (real np)
```

The overloaded *real* function is used here to “typecast” the natural number np into a real number of the same value. The auxiliary function *fiX* returns the process’ own clock reading if the reading of the other processes are more than Δ away.

```
fiX :: [(process ⇒ Clocktime), process, process] ⇒ Clocktime
fiX f p l ≡ if |f p - f l| <= Δ then (f l) else (f p)
```

We assume that both np and Δ are positive numbers.

```
axioms
  constants-ax: 0 <= Δ ∧ 0 < np
```

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 [Sha92]. The manual proof of the precision enhancement property uses eight auxiliary lemmas, three of them are about properties on summation 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 [Sha92]:

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

All the three properties concerning the convergence function are proved in Isabelle (the complete proof script can be found in [Bar06]). For the SMT prover, we have tested CVC-Lite, Yices and Ario and all of them successfully prove the selected lemmas in the following. Most of these lemmas concern the property of the *fiX* function, which is linear, and hence are in the scope of most SMT provers.

Precision enhancement lemma. We show that the convergence function *fiX* satisfies the precision enhancement condition in Section 3. The instance of the precision enhancement condition in the case of ICA protocol is stated in the following 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: $q \in C$ **and**

hrC: $r \in C$

shows $|cfni\ p\ f - cfni\ q\ g| \leq (real\ (card\ C) * (x + (\text{if } (y \leq \Delta) \text{ then } 0 \text{ else } y))) + real\ (card\ (\{..np'\} - C)) * (2 * \Delta + x + y) / real\ np$

The function *card* computes the cardinality of a set.

In a more familiar syntax, the above lemma states that

$$|cfni\ p\ f - cfni\ q\ g| \leq \frac{c \times (x + \text{if } y \leq \Delta \text{ then } 0 \text{ else } y) + k \times (2 \times \Delta + x + y)}{np}.$$

By unfolding the definition of *cfni*, and multiplying both sides of the above equation with np , we get the following inequality

$$|(\sum r. fiX\ f\ p\ r) - (\sum r. fiX\ g\ q\ r)| \leq c \times (x + \text{if } y \leq \Delta \text{ then } 0 \text{ else } y) + k \times (2 \times \Delta + x + y)$$

It is easy to see that the left term of this inequality is less or equal to

$$\sum r. |fiX\ f\ p\ r - fiX\ g\ q\ r|.$$

Note that this summation is over the set PR , so we can split it into two part:

$$\sum r \in C. |fiX\ f\ p\ r - fiX\ g\ q\ r| + \sum r \in PR \setminus C. |fiX\ f\ p\ r - fiX\ g\ q\ r|$$

The original equation is proved by establishing the following two facts:

$$\sum r \in C. |fiX\ f\ p\ r - fiX\ g\ q\ r| \leq c \times (x + \text{if } y \leq \Delta \text{ then } 0 \text{ else } y) \tag{1}$$

and

$$\sum r \in PR \setminus C. |fiX\ f\ p\ r - fiX\ g\ q\ r| \leq k \times (2 \times \Delta + x + y) \tag{2}$$

```

lemma abs-dif-fiX-bound:
assumes
  hbx:  $\forall l \in C. |f l - g l| \leq x$  and
  hby:  $\forall l \in C. \forall m \in C. |f l - f m| \leq y$  and
  hpC:  $p \in C$  and
  hqC:  $q \in C$ 
shows
   $|fiX f p r - fiX g q r| \leq 2 * \Delta + x + y$ 
proof-
have  $|(if |f p - f r| \leq \Delta then (f r) else (f p)) - (if |g q - g r| \leq \Delta then (g r) else (g q))| \leq 2 * \Delta + x + y$ 
proof-
  from hpC hbx have  $|f p - g p| \leq x$  by blast
  moreover
  from hqC hbx have  $|f q - g q| \leq x$  by blast
  moreover
  from hpC hqC hby have  $|f p - f q| \leq y$  by blast
  moreover
  from hpC hqC hby have  $|f q - f p| \leq y$  by blast
  moreover
  note constants-ax
  ultimately show ?thesis by smt
qed
thus ?thesis by (simp add: fiX-def)
qed

```

Fig. 7. A lemma used in the proof of precision enhancement of ICA.

Note that since c is the cardinality of C and k is the cardinality of $PR \setminus C$, to show that the equations (1) and (2) hold, it suffices to show the following:

$$|fiX f p r - fiX g q r| \leq (x + \text{if } y \leq \Delta \text{ then } 0 \text{ else } y)$$

$$|fiX f p r - fiX g q r| \leq (2 \times \Delta + x + y)$$

These equations are formalized as the following lemmas:

```

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
   $|fiX f p r - fiX g q r| \leq x + (\text{if } (y \leq \Delta) \text{ then } 0 \text{ else } y)$ 

```

```

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

```

Both lemmas are proved using the SMT oracle. We show the proof for the second lemma in Figure 7. The proof is done simply by instantiating the assumptions *hbx* and *hby* with processes p and q , and let the SMT prover do the rest. The proof of the first lemma is done similarly, but involves more instantiations.

Accuracy preservation The accuracy preservation theorem for ICA is formalized as follows

```

lemma accur-pres:
assumes
  hC:  $C \subseteq PR$  and

```

```

axioms
  constants-ax: 1 < np ∧ 2 * khl < np

constdefs
  PR :: process set
  PR ≡ {...np{}}
  declare PR-def[simp]

constdefs
  kmax :: (process ⇒ Clocktime) ⇒ process set ⇒ process set
  kmax f P ≡ SOME S. S ⊆ P ∧ card S = khl ∧
    (∀ i ∈ S. ∀ j ∈ (P-S). f j <= f i)
  kmin :: (process ⇒ Clocktime) ⇒ process set ⇒ process set
  kmin f P ≡ SOME S. S ⊆ P ∧ card S = khl ∧
    (∀ i ∈ S. ∀ j ∈ (P-S). f i <= f j)

  reduce :: (process ⇒ Clocktime) ⇒ process set ⇒ Clocktime set
  reduce f P ≡ f' (P - (kmax f P ∪ kmin f P))

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

```

Fig. 8. The convergence function of Lundelius-Lynch algorithm.

```

  hby: ∀ l ∈ C. ∀ m ∈ C. |f l - f m| <= x and
  hpC: p ∈ C and
  hqC: q ∈ C
shows |cfni p f - f q| <= (real (card C) * x + real (card {...np{}} - C)) * (x + Δ) / real np

```

By a similar analysis to the one in the precision enhancement lemma, we arrive at the following two key lemmas to the accuracy preservation property:

```

lemma bound-aux-C:
assumes
  hby: ∀ l ∈ C. ∀ m ∈ C. |f l - f m| <= x and
  hpC: p ∈ C and
  hqC: q ∈ C and
  hrC: r ∈ C
shows |fiX f p r - f q| <= x

```

```

lemma bound-aux:
assumes
  hby: ∀ l ∈ C. ∀ m ∈ C. |f l - f m| <= x and
  hpC: p ∈ C and
  hqC: q ∈ C
shows |fiX f p r - f q| <= x + Δ

```

These two lemmas are proved by using the SMT oracle, after some obvious steps of unfolding the definition of fiX and instantiations of the universal quantifiers in the assumptions.

The use of SMT oracle in verifying the ICA protocol has been the most successful among the case studies that we have done, in terms of the reduction of proof size and proof development time. The use of the SMT oracle subsumes most of the intermediate lemmas used in proving the precision enhancement and the accuracy preservations. This is mainly due to the linearity of the convergent function used in ICA.

5.2. 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 multiset of clock readings S , the algorithm first removes the k lowest and k highest values, and takes the midpoint of the remaining values. Its formalization in 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 8. 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 notation $f \cdot S$ 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 [LL84].

To prove the precision enhancement and the accuracy preservation, we use the bounding functions used to prove the same properties in [Min93]. 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 $\alpha y = y$.

Unlike the case with the ICA algorithm, the proof of the precision enhancement and the accuracy preservation in Lundelius-Lynch algorithm does not make much use of the SMT oracle. This is mainly because the proofs require reasoning about Hilbert's choice operator and non-trivial sets manipulation. Some parts of the proofs require statements about higher-order functions which have to be abstracted when we translate them to SMTLIB format. We illustrate below how we use the SMT oracle in proving the precision enhancement and the accuracy preservation properties.

Precision enhancement The precise statement of the precision enhancement is as follows.

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

The key lemma to proving the precision enhancement is the following:

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

This lemma, however, is not provable on its own. Several assumptions on the behaviour of the functions f and g are needed, that is, the assumptions *hbx*, *hby1* and *hby2* in the precision enhancement theorem above. There are two issues in proving the lemma using the SMT oracle. The first one is that the *reduce* function is not a first-order function, since it takes another function as argument. The more difficult one is how we instantiate the quantifiers in *hbx*, *hby1* and *hby2*. It may seem that the instances we need to consider are p and q . But it turns out that this is not sufficient. The precision enhancement theorem requires the following three intermediate lemmas:

lemma *uboundmax:*

assumes

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

hCk: $np \leq \text{card } C + khl$

shows

$\exists i \in C. Max (reduce f PR) \leq f i$

lemma *lboundmin:*

assumes

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

hCk: $np \leq \text{card } C + khl$

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–

⋮

let $?maxf = \text{Max}(\text{reduce } f\ PR)$
and $?minf = \text{Min}(\text{reduce } f\ PR)$ **and** $?maxg = \text{Max}(\text{reduce } g\ PR)$ **and** $?ming = \text{Min}(\text{reduce } g\ PR)$
let $?hx = \lambda l. |f l - g l| \leq x$
and $?hy1 = \lambda l. \lambda m. |f l - f m| \leq y$
and $?hy2 = \lambda l. \lambda m. |g l - g m| \leq y$

{

fix *max-red-f* :: *real*
fix *min-red-f* :: *real*
fix *max-red-g* :: *real*
fix *min-red-g* :: *real*

assume *maxf* : *max-red-f* = $?maxf$
and *minf* : *min-red-f* = $?minf$
and *maxg* : *max-red-g* = $?maxg$
and *ming* : *min-red-g* = $?ming$

have $|max-red-f + min-red-f + - max-red-g + - min-red-g| \leq y + 2 * x$

proof –

note *hC hCF hFn* **moreover**
from *uboundmax hC hCk maxf* **obtain** *pmaxf*
where *maxfC*: $pmaxf \in C$ **and** $max-red-f \leq f\ pmaxf$ **by** *blast* **moreover**
from *uboundmax hC hCk maxg* **obtain** *pmaxg*
where *maxgC*: $pmaxg \in C$ **and** $max-red-g \leq g\ pmaxg$ **by** *blast* **moreover**
from *lboundmin hC hCk minf* **obtain** *pminf*
where *minfC*: $pminf \in C$ **and** $f\ pminf \leq min-red-f$ **by** *blast* **moreover**
from *lboundmin hC hCk ming* **obtain** *pming*
where *mingC*: $pming \in C$ **and** $g\ pming \leq min-red-g$ **by** *blast* **moreover**
from *same-bound hC hCk hnk minf maxg* **obtain** *sbf*
where *sbfC*: $sbf \in C$ **and** $min-red-f \leq f\ sbf$ **and** $g\ sbf \leq max-red-g$ **by** *blast* **moreover**
from *same-bound hC hCk hnk ming maxf* **obtain** *sbfg*
where *sbfgC* : $sbfg \in C$ **and** $min-red-g \leq g\ sbfg$ **and** $f\ sbfg \leq max-red-f$ **by** *blast* **moreover**
from *hbx* **have** $?hx\ pmaxf$ **..** **moreover**

⋮

from *hby1 maxfC minfC* **have** $?hy1\ pmaxf\ pminf$ **by** *simp*

⋮

from *hby2 maxfC minfC* **have** $?hy2\ pmaxf\ pminf$ **by** *simp* **moreover**

⋮

ultimately show *?thesis* **by** *smt*

qed

}

hence $|?maxf + ?minf + - ?maxg + - ?ming| \leq y + 2 * x$ **by** *blast*

ultimately show *?thesis* **by** *simp*

qed

Fig. 9. A proof of the precision enhancement of Lundelius-Lynch protocol.

shows

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

lemma same-bound:

assumes

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

$hCk: np \leq \text{card } C + khl$ **and**

$hmk: 3 * khl < np$

shows

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

By Skolemizing the existential quantifiers, we obtain processes $pmaxf$, $pminf$, $pmaxg$, $pming$, $sbfg$ and $sbgf$ satisfying the following properties:

$$pmaxf \in C, \quad pminf \in C, \quad pmaxg \in C, \quad pming \in C, \quad sbfg \in C, \quad sbgf \in C$$

$$\text{Max}(\text{reduce } f \text{ PR}) \leq f pmaxf, \quad \text{Max}(\text{reduce } g \text{ PR}) \leq g pmaxg,$$

$$f pminf \leq \text{Min}(\text{reduce } f \text{ PR}), \quad g pming \leq \text{Min}(\text{reduce } g \text{ PR}),$$

$$\text{Min}(\text{reduce } f \text{ PR}) \leq f sbfg, \quad g sbfg \leq \text{Max}(\text{reduce } g \text{ PR}),$$

$$\text{Min}(\text{reduce } g \text{ PR}) \leq g sbgf, \quad f sbgf \leq \text{Max}(\text{reduce } f \text{ PR}).$$

We then proceed to instantiate the quantifiers in hbx , $hby1$ and $hby2$ with combinations of p , q , $pmaxf$, $pmaxg$, $pminf$, $pming$, $sbfg$ and $sbgf$. By accumulating these assumptions, we can now prove the precision enhancement property using the SMT oracle. A proof outline is given in Figure 9. Notice that in the proof we use proof blocks (see Section 2) to introduce new parameters to abstract the terms that contain higher-order functions.

Accuracy preservation The accuracy preservation theorem is proved in a similar way as the precision enhancement. In this case, however, fewer instantiations are needed. The complete proof, using the SMT oracle, is shown in Figure 10.

6. Conclusion and Future Work

The present paper mainly reports on case studies to investigate the integration of tools to obtain a platform suitable for formal verification of distributed algorithms. Our conclusion from these preliminary case studies is that the use of SMT provers does seem to help reducing both proof development time and proof size significantly. We think this is due to the large intersection between the problem domain that we pick and the kind of problems that the SMT provers solve. The work we present here should be useful as a basis for verifying more concrete clock synchronization protocols, such as the one used in the FlexRay protocol [Fle04], which is a variant of Lundelius-Lynch protocol. In general, the combination of proof assistants and SMT provers should be useful for other applications where arithmetics constraints and first-order reasoning are required, for instance, in the verification of the TCP/IP protocol done in [BFN⁺06]. Our current formalization of clock synchronization follows Shankar's formulation of clock axioms. However some of these axioms maybe too strong, that is, they may not correspond to the actual behaviours of physical clocks, as argued in [Min93]. There have been works in refining Shankar's formalization while maintaining its generality [Min93, SvH98]. We plan to study these works as well.

Our SMT interface does not handle quantifiers yet. However, this is not due to any technical difficulty in translating quantified formulas from Isabelle to SMTLIB. Rather, the SMT provers that we tested do not currently support quantified expressions in SMTLIB format, although they accept quantified formulas in their native syntax. For instance, as it has been reported in our preliminary work [BPT06], we actually manage to prove the accuracy preservation property of Lundelius-Lynch algorithm directly without instantiating the quantifiers first by translating it directly to CVC-Lite native syntax. The SMTLIB format is still evolving, and it is just a matter of time before a stable version will be supported by a large number of SMT provers. An immediate future work is to have quantifier instantiation done automatically, for the cases where the domain of the quantified variables is finite. This can be done by, e.g., writing a special tactic in Isabelle.

```

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-
{
  fix max-red :: real
  fix min-red :: real

  assume maxf : max-red = Max (reduce f PR)
  and minf : min-red = Min (reduce f PR)

  from reduce-not-empty
  have Min (reduce f PR)  $\leq$  Max (reduce f PR)
  by (auto simp add: reduce-def)

  from this maxf minf
  have le-red : min-red  $\leq$  max-red by simp

  have  $|(max-red + min-red)/2 - f q| \leq y$ 
  proof-
from hCF hFk have npleCk:  $np \leq \text{card } C + khl$  by arith
  moreover
from npleCk hC uboundmax maxf obtain pmaxf where
  pmaxfC:  $pmaxf \in C$  and max-red  $\leq$  f pmaxf by blast
  moreover
from npleCk hC lboundmin minf obtain pminf where
  pminfC:  $pminf \in C$  and f pminf  $\leq$  min-red by blast
  moreover
from pmaxfC pminfC hby have  $|f pminf - f pmaxf| \leq y$  by blast
  moreover
from pmaxfC pminfC hby have  $|f pmaxf - f pminf| \leq y$  by blast
  moreover
from pmaxfC hqC hby have  $|f q - f pmaxf| \leq y$  by blast
  moreover
from pmaxfC hqC hby have  $|f pmaxf - f q| \leq y$  by blast
  moreover
from pminfC hqC hby have  $|f q - f pminf| \leq y$  by blast
  moreover
from pminfC hqC hby have  $|f pminf - f q| \leq y$  by blast
  moreover
  note le-red
  ultimately show ?thesis by smt
  qed
}
thus ?thesis by (simp add: cfnl-def)

qed

```

Fig. 10. A proof of the accuracy preservation for Lundelius-Lynch protocol.

Proof reconstruction. An important question in combining different deductive tools is the overall consistency of the combination. There can be many sources of inconsistency, e.g., errors in the translation of syntax from one language to another, logical errors in translating types (e.g., translating integer variables to real variables), bugs in the external provers, etc. An ideal solution would be for the external tools to produce justification of their results. In fact, most of the SMT provers we tested, i.e., CVC-Lite, Yices and Ario, already have the functionality to produce proof traces. One major problem lies in the translation of these proof traces into Isabelle's proofs. Proofs produced by automated tools tend to be large in size, and they often use derived reduction rules which require many inference steps to reproduce in Isabelle. As a result, there might be significant performance cost in the translation. Proof reconstructions in combination of tools have recently been done for several tools, including SAT solvers [Web06, FMM⁺06], first-order the-

orem provers [MP04], and SMT solvers [MBG06, FMM⁺06]. We have experimented with proof-producing capabilities of some SMT provers. The tools seem to produce proofs of a “reasonable” size. For instance, the proof trace produced by Yices for the precision enhancement theorem for the Lundelius-Lynch algorithm has about 340 inference steps. Although this is quite large for human reader, it seems feasible for automated proof checkers. In [FMM⁺06], a compact proof trace is defined for the haRVey prover, for first-order logic with equality of uninterpreted function symbols. The proof trace consists of a list of *conflict clauses* containing only simple equations, which correspond to intermediate lemmas in the main proof. The idea is that the main proof is just a simple resolution on these clauses. The proof reconstruction on the Isabelle side is then reduced to proving each conflict clause and the final resolutions of these clauses. We plan to extend the proof reconstruction for haRVey, and possibly other SMT solvers, to include linear arithmetics. This work will include a definition of compact proof traces for linear arithmetics.

Acknowledgment. A major part of this work was done at LORIA, as part of the QSL Plateforme project. A preliminary version of the paper has appeared in [BPT06]. We are grateful for the comments and suggestions from the anonymous referees and members of the QSL project: Stephan Merz, Pascal Fontaine, Jean-Yves Marion, Antonia Balaa and Mauro-Javier Jaskelioff.

References

- [Bar06] Damian Barsotti. Instances of schneider’s generalized protocol of clock synchronization. In Gerwin Klein, Tobias Nipkow, and Lawrence Paulson, editors, *The Archive of Formal Proofs*. <http://afp.sf.net/entries/ClockSynchInst.shtml>, March 2006. Formal proof development.
- [BFN⁺06] Steve Bishop, Matthew Fairbairn, Michael Norrish, Peter Sewell, Michael Smith, and Keith Wansbrough. Engineering with logic: HOL specification and symbolic-evaluation testing for TCP implementations. In *POPL’06: Conference record of the 33rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 55–66, New York, NY, USA, 2006. ACM Press.
- [BKKS05] J. Botaschanjan, L. Kof, C. Kühnel, and M. Spichkova. Towards Verified Automotive Software. In ACM Press, editor, *Proceedings of the 2nd International ICSE workshop on Software*. ACM, New York, may 2005.
- [BPT06] Damian Barsotti, Leonor Prensa Nieto, and Alwen Tiu. Verification of clock synchronization algorithms: Experiments on a combination of deductive tools. *Electr. Notes Theor. Comput. Sci.*, 145:63–78, 2006.
- [CVC] CVC Lite. <http://chicory.stanford.edu/CVC/>.
- [DCB⁺03] Louise A. Dennis, Graham Collins, Richard Boulton, Konrad Slind, Graham Robinson, Mike Gordon, and Tom 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.
- [dM04] Leonardo de Moura. SAL: Tutorial. Computer Science Laboratory, SRI International, April 2004.
- [Fle04] FlexRay Consortium. *FlexRay Communications System Protocol Specification Version 2.0*, June 2004.
- [FMM⁺06] Pascal Fontaine, Jean-Yves Marion, Stephan Merz, Leonor Prensa Nieto, and Alwen Tiu. Expressiveness + automation + soundness: Towards combining smt solvers and interactive proof assistants. In Holger Hermanns and Jens Palsberg, editors, *TACAS*, volume 3920 of *Lecture Notes in Computer Science*, pages 167–181. Springer, 2006.
- [Isa] Isabelle home page. <http://isabelle.in.tum.de/>.
- [KS06] Christian Kühnel and Maria Spichkova. FlexRay and FTCom: Formale Spezifikation in FOCUS. Technical Report 10601, Technische Universität München, 2006.
- [LL84] Jennifer Lundelius and Nancy Lynch. A new fault-tolerant algorithm for clock synchronization. In *Proceedings of PODC ’84*, pages 75–88, New York, NY, USA, 1984. ACM Press.
- [LMS85] Leslie Lamport and P. M. Melliar-Smith. Synchronizing clocks in the presence of faults. *J. ACM*, 32(1):52–78, 1985.
- [MBG06] Sean McLaughlin, Clark Barrett, and Yeting Ge. Cooperating theorem provers: A case study combining hol-light and cvc lite. *Electr. Notes Theor. Comput. Sci.*, 144(2):43–51, 2006.
- [Min93] Paul S. Miner. Verification of fault-tolerant clock synchronization systems. NASA Technical Paper 3349, NASA Langley Research Center, November 1993.
- [MP04] Jia Meng and Lawrence C. Paulson. Experiments on supporting interactive proof using resolution. In David A. Basin and Michaël Rusinowitch, editors, *IJCAR*, volume 3097 of *Lecture Notes in Computer Science*, pages 372–384. Springer, 2004.
- [Nip02] Tobias Nipkow. Structured proofs in Isar/HOL. In Herman Geuvers and Freek Wiedijk, editors, *TYPES*, volume 2646 of *Lecture Notes in Computer Science*, pages 259–278. Springer, 2002.
- [RT05] Silvio Ranise and Cesare Tinelli. The SMT-LIB standard : Version 1.1, March 2005.
- [SBB⁺02] Jörg H. Siekmann, Christoph Benzmüller, Vladimir Brezhnev, Lassaad Cheikhrouhou, Armin Fiedler, Andreas Franke, Helmut Horacek, Michael Kohlhase, Andreas Meier, Erica Melis, Markus Moschner, Immanuel Normann, Martin Pollet, Volker Sorge, Carsten Ullrich, Claus-Peter Wirth, and Jürgen Zimmer. Proof development with OMEGA. In *CADE*, pages 144–149, 2002.
- [Sch87] Fred B. Schneider. Understanding protocols for Byzantine clock synchronization. Technical Report TR 87–859, Cornell University, 1987.

- [Sha92] Natarajan 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.
- [SvH98] Detlef Schwier and Friedrich von Henke. Mechanical verification of clock synchronization algorithms. In Anders P. Ravn and Hans Rischel, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, number 1486 in LNCS, pages 262–271. Springer, September 1998.
- [Tav04] Bertrand Tavernier. Calife: A generic graphical user interface for automata tools. *Electr. Notes Theor. Comput. Sci.*, 110:169–172, 2004.
- [Tiu05] Alwen Tiu. A formalization of a generalized clock synchronization protocol in Isabelle/HOL. In Gerwin Klein, Tobias Nipkow, and Lawrence Paulson, editors, *The Archive of Formal Proofs*. <http://afp.sf.net/entries/GenClock.shtml>, June 2005. Formal proof development.
- [Web06] Tjark Weber. Integrating a SAT solver with an LCF-style theorem prover. *Electr. Notes Theor. Comput. Sci.*, 144(2):67–78, 2006.
- [Yic] Yices home page. <http://fm.csl.sri.com/yices/>.