SMT-based Automation for Overwhelming Truth

David Baelde, Stéphanie Delaune, Stanislas Riou

SMT-based Automation for Overwhelming Truth

David Baelde
Univ Rennes, CNRS, IRISA, France

Stéphanie Delaune
Univ Rennes, CNRS, IRISA, France

Stanislas Riou
Univ Rennes, CNRS, IRISA, France

***Abstract*—Cryptographers are interested in showing that facts hold with overwhelming probability, i.e., a probability that grows fast enough to 1 wrt. some security parameter. It is thus natural to consider a formal logic where terms and formulas are interpreted as random variables, with a notion of validity based on overwhelming truth. In such a setting, one can postulate e.g. an axiom stating that the hash of two distinct adversarial terms do not collide, even though there is actually a negligible probability that an attacker finds such a collision. This results in a logic that is both fully formal and allows easy reasoning. However, the non-standard semantics of the logic makes it non-trivial to use common automation techniques. In this work, we show that it is actually possible to use classical reasoning tools, and more specifically SMT solvers. We develop this approach in practice in the setting of the Squirrel proof assistant, designing efficient encodings that leverage standard theories. We present benchmarks comparing our approach to existing automated reasoning techniques in Squirrel, and show how the new SMT-based tactics enable much shorter proof scripts.**

I. Introduction

Cryptographic protocols are omnipresent in our daily lives. From online banking and shopping transactions to email exchanges and social media interactions, cryptographic protocols play a prominent role in ensuring the confidentiality, integrity, and authenticity of transmitted data. They are hard to get right, and actually even well-studied cryptographic protocols such as Transport Layer Security (TLS) have been repeatedly found to be flawed [1].

Given the importance and inherent challenges of designing secure protocols, computer scientists have worked to provide solid mathematical foundations and tools for the computer-aided verification of these protocols. Since the early 1980s, two main approaches, known as *computational* and *symbolic*, have emerged to ground security analysis of protocols on rigorous mathematical foundations. Practitioners are now aware of the usefulness of formal methods. For example, the IETF has issued a call for the analysis of the EDHOC protocol [2], while Swiss authorities require formal guarantees regarding the electronic voting protocols they use [3, Annex 2.14].

The symbolic approach involves representing cryptographic messages as first-order terms, coupled with an equational theory that depicts attacker capabilities. Initially introduced in [4], this approach has evolved over time, giving rise to various symbolic models, and powerful automatic verification tools such as ProVerif and Tamarin. These tools have been successfully employed to analyze numerous protocols, e.g. the EMV standard for electronic payments [5] or Bluetooth protocols [6], leading to the discovery of various attacks. However, while these tools are useful to uncover attacks, the absence of an attack in the symbolic model is a weaker guarantee than in the cryptographer's standard security notion, based on the so-called computational model [7]. In such a model, messages are bitstrings, adversaries are arbitrary probabilistic polynomial-time (PTIME) Turing machines, and security properties on primitives and protocols are defined using games played by the attacker who has to be able to distinguish between two scenarios with a non-negligible probability. The main drawback of this more realistic model is that, even for small protocols, direct formal proofs are usually difficult, involving many details regarding probabilities and time complexity. When carried out on paper, such tedious proofs are error prone. Unfortunately, they are also very difficult to automate. Nonetheless, a few approaches to mechanizing computational proofs have emerged over the years. For instance, EasyCrypt [8] is a proof assistant that allows reasoning on cryptographic programs using probabilistic relational Hoare logic [9] (pRHL), with limited automation. Another successful system is CryptoVerif [10], which mechanizes the game-based proofs traditionally used by cryptographers, with a high level of automation.

In this paper, we consider a recent approach to proving protocols in the computational model, embodied in the proof assistant Squirrel [11], [12]:

https://squirrel-prover.github.io

It is based on the Computationally Complete Symbolic Attacker (CCSA) logic introduced by Bana and Comon [13], which relies on the symbolic setting of formal logic, but avoids the limitations of the symbolic models. The CCSA logic has been extended into a meta-logic to serve as a basis for the first version of Squirrel [11], before being generalized to a higher-order logic [12]. Notably, a formula in Squirrel's logic is valid when its is true with overwhelming probability (i.e., a probability that grows fast enough to 1 with the security parameter). Thanks to this non-classical notion of validity, the probabilistic aspects of proofs in the computational model remain implicit in Squirrel proofs. However, it is not obvious that classical automated reasoning techniques, such as those behind SMT solvers, are applicable to non-classical logics such as the one behind Squirrel.

The initial successes of the Squirrel system lie within a limited scope, in terms of the complexity of the protocols

This work received funding from the France 2030 program managed by the French National Research Agency under grant agreement No. ANR-22-PECY-0006.

and of the proofs that can actually be conducted. This is due, of course, to the fact that some cryptographic assumptions have not yet been integrated into this recent tool, but also to the fact that some non-cryptographic reasoning, e.g. invariants regarding counters, needs to be detailed in order to be validated step by step by the SQUIRREL proof assistant. To facilitate the development of proofs and enable larger ones, it is desirable to automate parts of the security proofs that do not involve cryptographic reasoning, so that the user can focus on the most interesting aspects. In fact, from the very first versions of the SQUIRREL tool, ad-hoc automated reasoning techniques have been implemented as tactics, but they were limited in scope, and difficult to extend. We aim to go further in this direction by leveraging existing automated theorem provers, particularly SMT solvers. More specifically, we shall rely on the WHY3 logic [14] and API to be able to benefit from the relative strengths of different provers.

Related work: SMT solvers are heavily used in most current program verification systems, e.g. WHY3 [15], DAFNY [16] or FRAMA-C [17] to cite only a few. In the field of cryptographic protocol verification, however, their usage is not as common.

Some cryptographic verification systems use SMT solvers to discharge proof obligations expressed in classical logic. The most prominent case in this category is EASYCRYPT [8]. At its core, EASYCRYPT is a proof assistant for classical higher-order logic. It relies on SMT solvers (through WHY3) to automatically discharge some goals, which requires a translation from classical higher-order to classical first-order logic. While the elimination of higher-order features is non-trivial, we stress that both logics are classical, unlike SQUIRREL's logic where validity is based on overwhelming truth. Cryptographic reasoning in EASYCRYPT is obtained via encodings of domain-specific logics (notably, pRHL) in the higher-order classical logic, and there is no specific support for SMT-based automation for these domain-specific logics. Other systems rely on a general-purpose logic to encode domain-specific logics, e.g. SSPROVE [18] and CRYPTHOL [19] respectively based on Rocq and Isabelle/HOL. Both Rocq and Isabelle/HOL support the use of SMT solvers to discharge subgoals [20], [21]. Again, this use of SMT solvers relies on translations that are not trivial but which stay within the realm of classical validity notions, whereas using SMT solvers in SQUIRREL requires a translation from overwhelming truth to classical truth.

Very recently, the standalone system CRYPTOVAMPIRE has been developed to fully automate proofs of trace properties in the CCSA logic [22]. CRYPTOVAMPIRE relies on a translation from a variant of SQUIRREL's logic and protocol modeling language to either VAMPIRE or plain SMT solvers. Unlike CRYPTOVAMPIRE, we only aim at automating some specific parts of the security proofs, leaving out cryptographic reasoning. As we shall see, this enables a more lightweight and flexible approach. Our work is also fully compatible with SQUIRREL's current logic, and integrated in the tool as a tactic, enabling proofs that mix our novel SMT-based reasoning with current (and future) tactics available in SQUIRREL. We discuss CRYPTOVAMPIRE in more detail at the end of the paper (see Section VIII).

Our contributions: The rest of the paper starts with an overview (Section II) illustrating the use of SQUIRREL on a simple stateful protocol, which is a typical target for SMT-based automation, after which the SQUIRREL and WHY3 logics are presented (Sections III and IV). We then introduce our contributions, which are three-fold:

- We develop in Section V a first translation from the SQUIRREL logic to the WHY3 logic, which allows to reduce the validity of SQUIRREL formulas to that of WHY3 formulas. This translation is very generic and treats all SQUIRREL constructs abstractly, but we explain how specific SQUIRREL assumptions can be incorporated as axioms in our implementation to allow for richer SMT-based reasoning.
- We then propose in Section VI a further translation, which we view as an optimization of our first result. We exploit standard SQUIRREL assumptions to reduce SQUIRREL validity to a WHY3 validity problem involving the theory of integers, in order to leverage theory-specific reasoning in SMT solvers.
- Both of our translations are fully integrated into the SQUIRREL tool, and made available as a new tactic. We present it in Section VII together with extensive experiments. We systematically evaluate the gains over prior automated reasoning in SQUIRREL and those associated to our optimized translation, and we show the significant benefits of our new tactic on a previous set of case studies on stateful protocols [23].

The SQUIRREL prover extended with the smt tactic, as well as all our case studies and benchmarks can be found as supplementary material of this report.

II. OVERVIEW

In this section, we give an overview of the SQUIRREL prover and in particular the smt tactic we have implemented. We use as a running example a toy protocol manipulating a counter.

Example 1: We consider a protocol inspired from a simple example given in [24] to illustrate the modeling of counters using PROVERIF. The protocol relies on a message authentication code (MAC), modeled as a keyed hash function, that is assumed to be unforgeable, and a global counter ctr that is incremented one by one. An agent executing this protocol first computes $m = \mathsf{h}(\mathsf{ctr}, k)$, and then increments the counter ctr before sending the hash value.

$$m := \mathsf{h}(\mathsf{ctr}, k); \quad \mathsf{ctr} := \mathsf{ctr} + 1; \ \text{send}\ m$$

We assume that this sequence of instructions can be executed an arbitrary number of times. Note that the key k and the counter ctr are shared between all the (honest) agents executing this protocol.

Listing 1 shows a formal description of the toy counter protocol in the input language of the SQUIRREL prover, which is close to the applied pi-calculus. In practice, the SQUIRREL tool translates this specification into a system of actions, each action representing a step of the protocol. Here, the protocol is rather simple and made of a unique action named A which can be repeated an arbitrary number of times. Therefore, this action is indexed by i: intuitively, $(\mathsf{A}\ i)$ represents the action performed in the i^{th} session of the role A.

```
type nat.
type hkey.

hash h.
name k : hkey.
channel c.

abstract one : nat
abstract succ : nat → nat
abstract inf: nat → nat → bool

axiom a1 (n:nat): inf n (succ n).
axiom a2 (n1,n2,n3:nat):
   (inf n1 n2) && (inf n2 n3) ⇒ (inf n1 n3).
axiom a3 (n1,n2:nat): n1 = n2 ⇒ ¬(inf n1 n2).

mutable ctr : nat = one.

process A = let m = h(ctr,k) in
            ctr := succ(ctr); out(c, m).

system !ᵢ A.
```

Listing 1. Toy counter protocol in SQUIRREL

Proving the security of this protocol requires reasoning on counter values. To that end, we axiomatize the ordering relation `inf` over `nat`, through the axioms given in Listing 1. We then express a security property of our protocol in SQUIRREL's language:

```
lemma reach : forall tau:timestamp,
  att(frame@tau) ≠ h(ctr@tau,k).
```

Listing 2. Reachability lemma in SQUIRREL

Here `frame@tau` is a *macro*, which stands for the sequence of messages emitted by the protocol until the timestamp `tau`. Our property expresses that, for any execution trace, and for any timepoint `tau`, the attacker is unable to compute the hash of the current counter. Note that we consider arbitrary attacker computations performed using past observables, represented by `frame@tau`. As the counter is systematically increased before sending out the hash of the current counter's value, the unforgeability of the keyed hash function h (seen as a MAC here), together with the axioms on `inf`, and `succ`, ensure rather clearly that the attacker is unable to produce the hash of the current counter's value. With the current version of the SQUIRREL prover, this reachability lemma necessitates a 15-lines proof script. It relies on two intermediate lemmas expressing that the value of the counter `ctr` strictly increases between two consecutive timepoints, and then between two timepoints such that $t \prec t'$.

```
lemma counterIncreasePred (t:timestamp):
  init ≺ t ⇒ (inf ctr@pred(t) ctr@t).

lemma counterIncrease (t,t':timestamp):
   t ≺ t' ⇒ (inf ctr@t ctr@t').
```

Listing 3. Intermediate lemmas in SQUIRREL

The proof script mostly involves reasoning about counters. We aim to improve the usability of the system by discharging such reasoning, that does not rely on cryptographic assumptions, to SMT solvers. Using the `smt` tactic which we have developed for that purpose, there is no need for the intermediate lemma `counterIncreasePred`, and the `counterIncrease` lemma can simply be proved with two tactics:

```
Proof. induction t; smt. Qed.
```

Listing 4. Proof of the counterIncrease lemma in SQUIRREL, using smt

The lemma `reach` can then be established quite easily:

```
Proof.
intro Eq; euf Eq.
use counterIncrease; smt.
Qed.
```

Listing 5. Proof of the reachability lemma in SQUIRREL, using smt

The first line of the proof script contains the cryptographic argument (Existential UnForgeability of h). Then the proof simply relies on the fact that the counter is increasing at each step, so that the protocol itself does not produce the hash value of the current value of the counter.

III. SQUIRREL'S LOGIC

We formally define SQUIRREL's *local* logic, which we seek to automate. More precisely, we consider (a fragment of) the higher-order CCSA logic of [12], which has replaced the earlier meta-logic underlying SQUIRREL [11], [23]. We shall not use the full generality of the higher-order CCSA logic, and will as a consequence simplify its presentation.

A. Syntax

The terms of the higher-order CCSA logic [12] are simply-typed λ-terms with recursive definitions. Here, we only consider terms that are *well-formed* as stated in Definition 1.

We consider a set of sorts $\mathcal{B}$ (base types) containing at least bool, a set of variables $\mathcal{X}$, and a set of function symbols $\mathcal{F}$. We require that each variable comes with its sort $s \in \mathcal{B}$, and each function symbol comes with a type of the form $s_1 \to \ldots \to s_n \to s$ for some $n \geq 0$ and sorts s and s_i.

Definition 1: The syntax of *well-formed* terms is given by the following grammar, where $x \in \mathcal{X}$, $s \in \mathcal{B}$ and $f \in \mathcal{F}$:

$$
\begin{array}{rcl}
t & ::= & x \\
& | & f\ t_1 \ldots t_n \\
& | & \mathsf{if}_s\ t_0\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2 \\
& | & \mathsf{true} : \mathsf{bool} \mid \mathsf{false} : \mathsf{bool} \\
& | & t_1 =_s t_2 \\
& | & t_1 \wedge t_2 \mid t_1 \vee t_2 \mid t_1 \Rightarrow t_2 \mid t_1 \Leftrightarrow t_2 \mid \neg t_1 \\
& | & \forall x{:}s.\ t \mid \exists x{:}s.\ t
\end{array}
$$

Note that the language does not feature polymorphism, hence the use of distinct symbols for distinct sorts, e.g. $=_s$, $\mathsf{if}_s\ \mathsf{then}\ \mathsf{else}$, ... The applications of function symbols have to respect the sorts, and thus when applying the conditional instruction, terms t_1 and t_2 must have the same sort s (with t_0 of sort bool). The same applies for $=_s$ and for the quantifiers. We will often use the letters ϕ and ψ to denote well-formed terms of sort bool, since these terms will play the role of formulas. We sometimes do not indicate sorts, and write e.g. $=$ (instead of $=_s$) when they are clear from the context.

Example 2: We reformulate the concrete syntax used in the previous section as well-formed terms. We shall use, among others, the sorts msg and nat, in addition to bool. We assume two function symbols:

$$\mathsf{succ} : \mathsf{nat} \to \mathsf{nat} \qquad \mathsf{inf} : \mathsf{nat} \to \mathsf{nat} \to \mathsf{bool}$$

The symbol succ represents the successor of a term of sort nat, whereas inf is used to compare two terms of sort nat. The following well-formed term expresses the fact that any term x of sort nat is less than its successor $(\mathsf{succ}\ x)$ which corresponds to axiom `a1` in Listing 1.

$$\forall x{:}\mathsf{nat}.\ (\mathsf{inf}\ x\ (\mathsf{succ}\ x))$$

Well-formed terms are obtained from the general terms of higher-order CCSA logic by:

(i) assuming some specific function symbols (propositional connectives, quantifiers, etc.) which are indeed taken as builtins in SQUIRREL;

(ii) restricting to terms of base type, and restricting quantification to base types.

More about the link between the language presented here, and the one given in [12] can be found in Appendix A.

Note that, in the meta-logic of [11], [23], quantification is restricted to *finite* types (i.e. types which are interpreted as finite sets) to ensure that formulas are computable but this is no longer the case in [12]. Thus, we do not have such a limitation here, and we can quantify e.g. over type msg which is typically used to model bitstrings of arbitrary length.

B. Semantics

Well-formed terms of sort s will be interpreted as families of random variables over the interpretation of s: for each value of the security parameter $\eta \in \mathbb{N}$, the term can be seen as a random variable, taking different values depending on the random sampling. This allows to model, e.g., keys as random bitstrings of length η.

Formally, a model $\mathcal{M}$ defines, for each sort $s \in \mathcal{B}$, and for each value of the security parameter η, an interpretation domain $[\![s]\!]^{\mathsf{s}}_{\mathcal{M},\eta}$. We require that bool is given its natural interpretation in all models: $[\![\mathsf{bool}]\!]^{\mathsf{s}}_{\mathcal{M},\eta} = \{0,1\}$ for all $\mathcal{M}$ and η. Parameterizing the interpretation in η is useful e.g. to model key spaces that vary with the security parameter. This interpretation is lifted to arbitrary types by letting

$$[\![\tau_1 \to \tau_2]\!]^{\mathsf{s}}_{\mathcal{M},\eta} = [\![\tau_1]\!]^{\mathsf{s}}_{\mathcal{M},\eta} \to [\![\tau_2]\!]^{\mathsf{s}}_{\mathcal{M},\eta}.$$

A model must also define, for each value η of the security parameter, a finite set $\mathbb{T}_{\mathcal{M},\eta}$ of bitstrings of the same length, whose elements are called random tapes and noted using the letter ρ. These random tapes are used to obtain probabilistic values in the semantics. For example, different keys used in a protocol would be represented by different *name* symbols, interpreted as disjoint fragments of the random tape of length η.

An η-indexed family of random variables over type τ is then defined as a function f such that, for any $\eta \in \mathbb{N}$, we have that:

$$f(\eta) : \mathbb{T}_{\mathcal{M},\eta} \to [\![\tau]\!]^{\mathsf{s}}_{\mathcal{M},\eta}.$$

We let $\mathbb{RV}_{\mathcal{M}}(\tau)$ be the set of such families of random variables. When $f \in \mathbb{RV}_{\mathcal{M}}(\tau)$, we write $f(\eta,\rho)$ rather than $f(\eta)(\rho)$. Finally, we say that $\mathcal{M}$ is a model wrt. $(\mathcal{X},\mathcal{F})$ when it provides:

- for each $x : s$ in $\mathcal{X}$, an interpretation $\mathcal{M}(x) \in \mathbb{RV}_{\mathcal{M}}(s)$;
- for each $f : s_1 \to \ldots \to s_n \to s$ in $\mathcal{F}$, an interpretation $\mathcal{M}(f) \in \mathbb{RV}_{\mathcal{M}}(s_1 \to \ldots \to s_n \to s)$.

Given a model $\mathcal{M}$ wrt. $(\mathcal{X},\mathcal{F})$, and $V \in \mathbb{RV}_{\mathcal{M}}(s)$, we define the model $\mathcal{M}[x \mapsto V]$ wrt. $(\mathcal{X} \cup \{x : s\},\mathcal{F})$ which extends $\mathcal{M}$ by mapping x to V, i.e. $(\mathcal{M}[x \mapsto V])(x) = V$.

Definition 2: For any well-formed term t of sort s, and for any model $\mathcal{M}$ wrt. $(\mathcal{X},\mathcal{F})$, we define the interpretation $[\![t]\!]^{\mathsf{s}}_{\mathcal{M}} \in \mathbb{RV}_{\mathcal{M}}(s)$ according to the equations of Fig. 1.

Example 3: In a model $\mathcal{M}$ for our running example, the types nat and msg would typically be interpreted as the set of bitstrings, independently of η:

$$[\![\mathsf{nat}]\!]^{\mathsf{s}}_{\mathcal{M},\eta} = [\![\mathsf{msg}]\!]^{\mathsf{s}}_{\mathcal{M},\eta} = \{0,1\}^* \text{ for all } \eta.$$

However, we might have $[\![\mathsf{hkey}]\!]^{\mathsf{s}}_{\mathcal{M},\eta} = \{0,1\}^\eta$ to model keys used to hash. At the level of terms, if the function symbol

$$\mathsf{h} : \mathsf{nat} \to \mathsf{hkey} \to \mathsf{msg}$$

is meant to model a keyed hash function, we would only consider models where $\mathcal{M}(\mathsf{h})(\eta,\rho)$ is, given a value of η, the same function of $\{0,1\}^* \to \{0,1\}^\eta \to \{0,1\}^*$ for all ρ. In other words, the computation of a hash does not involve random samplings. The drawing of the key passed to that function, however, would be represented by a term k such that $[\![k]\!]^{\mathsf{s}}_{\mathcal{M}}(\eta,\rho)$ is some specific portion of length η of the random tape ρ.

In the rest of the paper, we omit the indication of $\mathcal{M}$ in interpretations when it is irrelevant or can be inferred.

$$[\![x]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho) = \mathcal{M}(x)(\eta,\rho) \qquad [\![f\ t_1 \dots t_n]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho) = F([\![t_1]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho),\dots,[\![t_n]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho)) \text{ where } F = \mathcal{M}(f)(\eta,\rho)$$

$$[\![\mathsf{if}\ t_1\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho) = [\![t_2]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho) \text{ when } [\![t_1]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho) = 1,\ [\![t_3]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho) \text{ otherwise}$$

$$[\![t_1 = t_2]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho) = 1 \text{ iff } [\![t_1]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho) = [\![t_2]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho) \qquad [\![t_1 \wedge t_2]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho) = 1 \text{ iff } [\![t_1]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho) = [\![t_2]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho) = 1$$

$$[\![\forall x : s.\ \phi]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho) = 1 \text{ iff } [\![\phi]\!]^{\mathsf{S}}_{\mathcal{M}[x \mapsto \mathbb{1}_a^{\eta,\rho}]}(\eta,\rho) = 1 \text{ for all } a \in [\![s]\!]^{\mathsf{S}}_{\mathcal{M},\eta}$$

In the last equation, $\mathbb{1}_a^{\eta,\rho} \in \mathbb{RV}_{\mathcal{M}}(\tau)$ is such that $\mathbb{1}_a^{\eta,\rho}(\eta,\rho) = a$ and $\mathbb{1}_a^{\eta,\rho}(\eta',\rho')$ takes an irrelevant value when $(\eta',\rho') \neq (\eta,\rho)$. Equations defining the interpretation of other logical constructs are similar.

Fig. 1. Semantics of terms

C. Local logic

Well-formed *formulas* are well-formed terms of type bool. They form a subset of the so-called *local* formulas in the higher-order CCSA logic. The adjective local comes from the fact that the semantics $[\![t]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho)$ only depends on the semantics of subterms t' of t for the same values of η and ρ, and is used in the higher-order CCSA logic when a clear distinction with *global formulas* is needed. However, that global logic is not considered in this paper. We define next the notion of validity for SQUIRREL's local logic which corresponds to cryptographic truth. This notion of validity applies, in particular to our well-formed terms of type bool.

Definition 3: A local formula ϕ wrt. $(\mathcal{X},\mathcal{F})$ is said to be *satisfied* in a model $\mathcal{M}$ wrt. $(\mathcal{X},\mathcal{F})$ when $[\![\phi]\!]^{\mathsf{S}}_{\mathcal{M}}$ is true with overwhelming probability, i.e. the function

$$\eta \mapsto 1 - \Pr_{\rho \in \mathbb{T}_{\mathcal{M},\eta}}([\![\phi]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho) = 1)$$

is asymptotically smaller than the inverse of any positive polynomial.

In practice, we seek to show that a formula is satisfied in all models from a given class, specified through a theory $\mathcal{T}$.

Definition 4: Let $\mathcal{T}$ be a set of local formulas, and ϕ be a local formula, all wrt. the same $(\mathcal{X},\mathcal{F})$. We say that ϕ is $\mathcal{T}$-valid, noted $\models^{\mathsf{S}}_{\mathcal{T}} \phi$, when any model $\mathcal{M}$ satisfying all formulas of $\mathcal{T}$ also satisfies ϕ. A formula ϕ is said to be valid, denoted $\models^{\mathsf{S}} \phi$, when it is $\emptyset$-valid.

Despite the non-standard semantics of SQUIRREL's logic, some interesting results have been established. For instance, it has been shown in [12, Rule L.Localise], that $\mathcal{T}$-validity can be reduced to plain validity when $\mathcal{T}$ is finite: $\models^{\mathsf{S}}_{\mathcal{T}} \phi$ is a consequence of $\models^{\mathsf{S}} (\bigwedge_{\psi \in \mathcal{T}} \psi) \Rightarrow \phi$. This is not trivial, as $\mathcal{T}$-validity states that if axioms hold with overwhelming probability then so does ϕ, while the reformulated validity states that the implication taken as a whole holds with overwhelming probability.

D. SQUIRREL specifics

SQUIRREL makes use of the general higher-order CCSA logic, specializing it to constrain the semantics of some types and function symbols (which may be builtins or user-declared) in the considered class of models. Our translation to classical first-order logic is largely independent of these details, which can be simply handled by translating a SQUIRREL theory describing the considered class of models to a WHY3 theory. We thus only sketch below the main assumptions used in SQUIRREL to model cryptographic protocols, insisting on the notion of trace model which will play a more profound role in Section VI. Specificities related to the way names and cryptographic primitives are modeled are available in Appendix A-B.

1) Timestamps: In order to reason about protocols and their execution traces, SQUIRREL makes a specific use of the index and timestamp sorts, equipped with several builtin functions. The interpretations of these sorts and functions are assumed to be *constant*, i.e. independent of η and ρ, but this is not relevant for this work.

More importantly, SQUIRREL restricts to models that interpret timestamp as a finite set. It is also assumed that the builtin happens : timestamp → bool is interpreted as a function which returns true on all timestamps but one — that special timestamp is sometimes named undef. Intuitively, the happens predicate is used to represent the timestamps appearing in the trace [23]. Further, the function symbol

$$\preceq : \mathsf{timestamp} \to \mathsf{timestamp} \to \mathsf{bool}$$

(used in infix notation) is interpreted as a total order over the timestamps that happen. Finally, function symbols init : timestamp and pred : timestamp → timestamp must represent the beginning of every trace, and the predecessor (according to the order $\preceq$) of a timestamp that happens. The axioms listed in Fig. 2 are the protocol-independent axioms expressible in first-order logic that hold in the considered class of models. Note that this excludes the axiom expressing the finiteness of timestamp.

We then need to concretize this abstract notion of trace to correspond to the protocol under study. To do so, a protocol is modeled as a set of *actions* $\mathcal{A} = \{\mathsf{A},\mathsf{B},\dots\}$, each symbol $\mathsf{A} \in \mathcal{A}$ having an arity $\mathsf{ar}(\mathsf{A})$ arising from considering several sessions.

$$\mathsf{A} : \mathsf{index} \to \dots \to \mathsf{index} \to \mathsf{timestamp}$$

We restrict to models where all timestamps that happen, except init, correspond to a unique action identifier, i.e. we

$$(\mathsf{happens\ init}) \quad (\mathsf{ax}_1)$$
$$(\mathsf{happens}\ t) \Rightarrow \mathsf{init} \preceq t \quad (\mathsf{ax}_2)$$
$$(\mathsf{happens}\ t) \Rightarrow (t = \mathsf{init}) \vee (\mathsf{happens}\ (\mathsf{pred}\ t)) \quad (\mathsf{ax}_3)$$
$$(\mathsf{happens}\ t_1) \vee (\mathsf{happens}\ t_2) \vee t_1 = t_2 \quad (\mathsf{ax}_4)$$
$$t_1 \preceq t_2 \wedge t_2 \preceq t_3 \Rightarrow t_1 \preceq t_3 \quad (\mathsf{ax}_5)$$
$$t_1 \preceq t_2 \wedge t_2 \preceq t_1 \Rightarrow t_1 = t_2 \quad (\mathsf{ax}_6)$$
$$(\mathsf{happens}\ t_1) \wedge (\mathsf{happens}\ t_2) \Leftrightarrow t_1 \preceq t_2 \vee t_2 \preceq t_1 \quad (\mathsf{ax}_7)$$
$$(\mathsf{happens}\ (\mathsf{pred}\ t)) \Rightarrow (\mathsf{pred}\ t) \preceq t \quad (\mathsf{ax}_8)$$
$$(\mathsf{happens}\ t) \Rightarrow \neg\, (\mathsf{pred}\ t) = t \quad (\mathsf{ax}_9)$$
$$(\mathsf{happens}\ (\mathsf{pred}\ t)) \Rightarrow (\mathsf{happens}\ t) \quad (\mathsf{ax}_{10})$$
$$(\mathsf{happens}\ (\mathsf{pred}\ t_1)) \Rightarrow (\mathsf{happens}\ t_2) \Rightarrow t_2 \preceq (\mathsf{pred}\ t_1) \vee t_1 \preceq t_2 \quad (\mathsf{ax}_{11})$$

Fig. 2. SQUIRREL axioms regarding timestamps. Variables t, t_1, t_2 of sort timestamp are implicitly assumed to be universally quantified.

restrict to models satisfying the following axiom:

$$\forall t.\ (\mathsf{happens}\ t) \Rightarrow t = \mathsf{init} \vee \bigvee_{\mathsf{A} \in \mathcal{A}} \exists \vec{i}.\ \ t = (\mathsf{A}\ \vec{i}) \quad (1)$$

Considering any $\mathsf{A}, \mathsf{B} \in \mathcal{A}$ such that $\mathsf{A} \neq \mathsf{B}$, we also require:

$$\forall \vec{i}, \vec{j}.\ \mathsf{happens}\ (\mathsf{A}\ \vec{i}) \Rightarrow (\mathsf{A}\ \vec{i}) = (\mathsf{A}\ \vec{j}) \Rightarrow \vec{i} = \vec{j} \quad (2)$$

$$\forall \vec{i}, \vec{j}.\ \mathsf{happens}\ (\mathsf{A}\ \vec{i}) \Rightarrow (\mathsf{A}\ \vec{i}) \neq (\mathsf{B}\ \vec{j}) \quad (3)$$

2) Macros: Finally, the semantics of protocol actions is defined through special function symbols, notably:

$$\begin{aligned} \mathsf{input}, \mathsf{output} &: \mathsf{timestamp} \to \mathsf{msg} \\ \mathsf{cond}, \mathsf{exec} &: \mathsf{timestamp} \to \mathsf{bool} \\ \mathsf{frame} &: \mathsf{timestamp} \to \mathsf{msg} \end{aligned}$$

Intuitively, the first three represent the input, output and executability condition of the action at the given timestamp (macro cond), provided that it happens. Then, exec is the conjunction of all executability conditions up to the given timestamp, and frame accumulates all protocol observables (i.e., accumulated executability conditions and outputs as long as executability holds) until a given timestamp. For protocols using mutable memory cells, we introduce more symbols to model the contents of each cell at each timestamp.

All these symbols, called *macros* in the former CCSA meta-logic [23], are defined using the recursive definition mechanism of [12]. For our purposes, we can equivalently express their semantics axiomatically. Some axioms are independent of the protocol. For instance, inputs are modeled as arbitrary adversarial computations using the attacker's knowledge:

$$\forall t.\ \big((\mathsf{happens}\, t) \wedge t \neq \mathsf{init}\big) \Rightarrow (\mathsf{input}\, t) = (\mathsf{att}\, (\mathsf{frame}\, (\mathsf{pred}\, t))) \quad (4)$$

Example 4: For our running example, we have a single action symbol A of arity 1, and we use a macro ctr : timestamp $\to$ msg to model the value of the mutable counter at some timestamp. We then axiomatize the protocol semantics as follows:

$$(\mathsf{ctr}\ \mathsf{init}) = \mathsf{one}$$
$$\forall i.\ (\mathsf{happens}\ (\mathsf{A}\, i)) \Rightarrow (\mathsf{ctr}\ (\mathsf{A}\, i)) = (\mathsf{succ}\ (\mathsf{ctr}\ (\mathsf{pred}\ (\mathsf{A}\, i))))$$
$$\forall i.\ (\mathsf{happens}\ (\mathsf{A}\, i)) \Rightarrow (\mathsf{output}\ (\mathsf{A}\, i)) = (\mathsf{h}\ (\mathsf{ctr}\ (\mathsf{pred}\ (\mathsf{A}\, i)))\ \mathsf{k})$$

IV. WHY3 LANGUAGE

We use the WHY3 language in order to target different SMT solvers. We recall in this section the syntax and semantics of WHY3 [14], leaving out the parts that we do not use, e.g. polymorphism and sort symbols with non-zero arity.

A. Syntax

We consider a set Σ_S of *sort symbols* of arity 0, and we assume that it contains at least the sort symbols bool/0, and int/0. The latter will be useful when considering our optimized translation.

We consider a set Σ_F of *function symbols*, as well as a set Σ_P of *predicate symbols*. Function symbols are declared in Σ_F with their arity, in the form $f(s_1, \ldots, s_n) : s_0$. Similarly, predicate symbols are declared in Σ_P in the form $p(s_1, \ldots, s_n)$. In both cases, $n \geq 0$, and each s_i is a sort in Σ_S. We assume that the set Σ_F contains at least the symbols True : bool and False : bool, and that Σ_P contains at least the equality predicates $=_s(s, s)$ for all $s \in \Sigma_S$. Finally, we also consider a set of variables $\mathcal{X}$, where each variable is given with its sort.

The standard semantics of WHY3 relies on a polymorphic equality. We slightly depart from this presentation to ease reasoning on our translations.

WHY3 terms and formulas are defined as follows:

$$\begin{aligned} t, t_1, \ldots, t_n ::= \ & x \text{ with } x \in \mathcal{X} \\ \mid \ & f(t_1, \ldots, t_n) \text{ with } f \in \Sigma_F \text{ of arity } n \\ \mid \ & \mathsf{if}\ \phi\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2 \end{aligned}$$

$$\begin{aligned} \phi, \phi_1, \ldots, \phi_n ::= \ & p(t_1, \ldots, t_n) \text{ with } p \in \Sigma_P \text{ of arity } n \\ \mid \ & \forall x{:}s.\phi \mid \exists x{:}s.\phi \quad \text{with } s \in \Sigma_S \\ \mid \ & \phi_1 \diamond \phi_2 \text{ with } \diamond \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\} \\ \mid \ & \neg\phi \mid \mathsf{true} \mid \mathsf{false} \\ \mid \ & \mathsf{if}\ \phi\ \mathsf{then}\ \phi_1\ \mathsf{else}\ \phi_2 \end{aligned}$$

We only consider well-typed terms, where sorts are respected and then and else branches of conditionals have the same type.

B. Semantics

To give a semantics to the WHY3 terms and formulas, we have to fix an interpretation that gives a meaning to the sorts, function and predicate symbols, as well as to the variables. Formally, given a signature $(\Sigma_S, \Sigma_F, \Sigma_P)$ and a set of variables $\mathcal{X}$, a WHY3 *interpretation* $\mathcal{I}$ associates:

- the domain $[\![\mathsf{bool}]\!]^{\mathsf{W}}_{\mathcal{I}} = \{\top, \bot\}$ to the sort bool;
- the domain $[\![\mathsf{int}]\!]^{\mathsf{W}}_{\mathcal{I}} = \mathbb{Z}$ to the sort int;
- otherwise a non-empty domain denoted $[\![s]\!]^{\mathsf{W}}_{\mathcal{I}}$ to each sort $s \in \Sigma_S$;

- a function, denoted $[\![f]\!]^{\mathsf{W}}_{\mathcal{I}}$, of type
$$[\![s_1]\!]^{\mathsf{W}}_{\mathcal{I}} \times \ldots \times [\![s_n]\!]^{\mathsf{W}}_{\mathcal{I}} \to [\![s_0]\!]^{\mathsf{W}}_{\mathcal{I}}$$
for each function symbol $f(s_1, \ldots, s_n) : s_0 \in \Sigma_F$, such that $[\![\mathsf{True}]\!]^{\mathsf{W}}_{\mathcal{I}} = \top$, and $[\![\mathsf{False}]\!]^{\mathsf{W}}_{\mathcal{I}} = \bot$;
- a function, denoted $[\![p]\!]^{\mathsf{W}}_{\mathcal{I}}$, of type
$$[\![s_1]\!]^{\mathsf{W}}_{\mathcal{I}} \times \ldots \times [\![s_n]\!]^{\mathsf{W}}_{\mathcal{I}} \to [\![\mathsf{bool}]\!]^{\mathsf{W}}_{\mathcal{I}}$$
for each predicate symbol $p(s_1, \ldots, s_n) \in \Sigma_P$, such that $[\![=_s]\!]^{\mathsf{W}}_{\mathcal{I}}$ is interpreted as the equality over $[\![s]\!]^{\mathsf{W}}_{\mathcal{I}}$ for each $s \in \Sigma_S$;
- an element $\mathcal{I}(x) \in [\![s]\!]^{\mathsf{W}}_{\mathcal{I}}$ for each variable $x : s$ in $\mathcal{X}$.

We write $\mathcal{I}[x \mapsto a]$ for the interpretation obtained by updating $\mathcal{I}$ with $\mathcal{I}(x) = a$ assuming that x and a have the same sort.

Once the interpretation $\mathcal{I}$ is fixed, we can extend it to give an interpretation to terms and formulas as defined in Fig. 3 and Fig. 4.

$$[\![x]\!]^{\mathsf{W}}_{\mathcal{I}} = \mathcal{I}(x)$$
$$[\![f(t_1, \ldots, t_n)]\!]^{\mathsf{W}}_{\mathcal{I}} = [\![f]\!]^{\mathsf{W}}_{\mathcal{I}}([\![t_1]\!]^{\mathsf{W}}_{\mathcal{I}}, \ldots, [\![t_n]\!]^{\mathsf{W}}_{\mathcal{I}})$$
$$[\![\mathsf{if}\ \phi\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2]\!]^{\mathsf{W}}_{\mathcal{I}} = \begin{cases} [\![t_1]\!]^{\mathsf{W}}_{\mathcal{I}} & \text{when } [\![\phi]\!]^{\mathsf{W}}_{\mathcal{I}} = \top \\ [\![t_2]\!]^{\mathsf{W}}_{\mathcal{I}} & \text{otherwise} \end{cases}$$

Fig. 3. WHY3 semantics for terms

$$[\![p(t_1, \ldots, t_n)]\!]^{\mathsf{W}}_{\mathcal{I}} = [\![p]\!]^{\mathsf{W}}_{\mathcal{I}}([\![t_1]\!]^{\mathsf{W}}_{\mathcal{I}}, \ldots, [\![t_n]\!]^{\mathsf{W}}_{\mathcal{I}})$$
$$[\![\forall x{:}s.\phi]\!]^{\mathsf{W}}_{\mathcal{I}} = \begin{cases} \top \text{ when } [\![\phi]\!]^{\mathsf{W}}_{\mathcal{I}[x\mapsto a]} = \top \text{ for all } a \in [\![s]\!]^{\mathsf{W}}_{\mathcal{I}} \\ \bot \text{ otherwise} \end{cases}$$
$$[\![\exists x{:}s.\phi]\!]^{\mathsf{W}}_{\mathcal{I}} = \begin{cases} \top \text{ when } [\![\phi]\!]^{\mathsf{W}}_{\mathcal{I}[x\mapsto a]} = \top \text{ for some } a \in [\![s]\!]^{\mathsf{W}}_{\mathcal{I}} \\ \bot \text{ otherwise} \end{cases}$$
$$[\![\phi_1 \diamond \phi_2]\!]^{\mathsf{W}}_{\mathcal{I}} = [\![\phi_1]\!]^{\mathsf{W}}_{\mathcal{I}} \diamond [\![\phi_2]\!]^{\mathsf{W}}_{\mathcal{I}} \text{ with } \diamond \in \{\vee, \wedge, \Rightarrow, \Leftrightarrow\}$$
$$[\![\neg\phi]\!]^{\mathsf{W}}_{\mathcal{I}} = \neg[\![\phi]\!]^{\mathsf{W}}_{\mathcal{I}} \qquad [\![\mathsf{true}]\!]^{\mathsf{W}}_{\mathcal{I}} = \top \qquad [\![\mathsf{false}]\!]^{\mathsf{W}}_{\mathcal{I}} = \bot$$
$$[\![\mathsf{if}\ \phi\ \mathsf{then}\ \phi_1\ \mathsf{else}\ \phi_2]\!]^{\mathsf{W}}_{\mathcal{I}} = \begin{cases} [\![\phi_1]\!]^{\mathsf{W}}_{\mathcal{I}} & \text{when } [\![\phi]\!]^{\mathsf{W}}_{\mathcal{I}} = \top \\ [\![\phi_2]\!]^{\mathsf{W}}_{\mathcal{I}} & \text{otherwise} \end{cases}$$

Fig. 4. WHY3 semantics for formulas

We say that an interpretation $\mathcal{I}$ satisfies a formula ϕ when $[\![\phi]\!]^{\mathsf{W}}_{\mathcal{I}} = \top$. A formula ϕ is *valid*, noted $\models^{\mathsf{W}} \phi$, when all interpretations satisfy ϕ. Lastly, given a finite set $\mathcal{T}$ of formulas, the formula ϕ is $\mathcal{T}$-valid, noted $\models^{\mathsf{W}}_{\mathcal{T}} \phi$, when $\models^{\mathsf{W}} (\bigwedge_{\psi \in \mathcal{T}} \psi) \Rightarrow \phi$.

V. CORE TRANSLATION

We now present our first translation from SQUIRREL to WHY3. This first translation is, syntactically speaking, rather straightforward, and it does not rely on any assumption on the considered class of SQUIRREL models. We will optimize it next, for the particular class of models used in SQUIRREL to represent protocols, by leveraging the theory of integers.

The translation is described in Section V-A, and its soundness is established in Section V-B. We then discuss how we use it within SQUIRREL in Section V-C.

A. Translation

Our translation is parameterized by a set $\mathcal{X}$ of typed variables, a SQUIRREL signature $(\mathcal{B}, \mathcal{F})$, and a WHY3 signature $(\Sigma_S, \Sigma_F, \Sigma_P)$ such that:

- $\Sigma_S = \mathcal{B} \cup \{\mathsf{int}/0\}$;
- $\{=_s \mid s \in \Sigma_S\} \subseteq \Sigma_P \subseteq \{p : s_1 \to \ldots \to s_n \to \mathsf{bool} \in \mathcal{F}\}$;
- $\Sigma_F = \mathcal{F} \setminus \Sigma_P$.

We use the same set of variables $\mathcal{X}$ in SQUIRREL and WHY3, and the same sorts, except for the addition of int to Σ_S which is required as a WHY3 builtin. Note that there is some flexibility in the choice of Σ_P: a function symbol returning a boolean in $\mathcal{F}$ can be considered as a WHY3 predicate (in Σ_P) or as a WHY3 function (in Σ_F).

Example 5: We consider the SQUIRREL signature needed for modeling the running example of Section II, which notably contains the following declarations:

h :	nat → hkey → msg	one :	nat
A :	index → timestamp	succ :	nat → nat
ctr :	timestamp → nat	inf :	nat → nat → bool

When translating this SQUIRREL signature into a WHY3 signature, we have to split $\mathcal{F}$ in Σ_F and Σ_P, and inf can actually be put in one set or another. This choice will lead to slightly different translations, but will not affect soundness.

We now define our translations from the well-formed terms of SQUIRREL to WHY3 terms and formulas. More precisely, we will define three transformations:

1) $\mathsf{W}_{\mathbf{f}}$ from SQUIRREL terms of type bool to WHY3 formulas;
2) $\mathsf{W}_{\mathbf{t}}$ from SQUIRREL terms of other types to WHY3 terms;
3) $\mathsf{W}'_{\mathbf{t}}$ from arbitrary SQUIRREL terms to WHY3 terms.

1) *Transformation* $\mathsf{W}_{\mathbf{f}}$. For any SQUIRREL term t of type bool, $\mathsf{W}_{\mathbf{f}}(\phi)$ is defined as the following WHY3 formula:

- $\mathsf{W}_{\mathbf{f}}(\phi_1) \diamond \mathsf{W}_{\mathbf{f}}(\phi_2)$ when ϕ is of the form $\phi_1 \diamond \phi_2$ with $\diamond \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$;
- $\neg \mathsf{W}_{\mathbf{f}}(\phi_0)$ when ϕ is $\neg\phi_0$;
- $\forall x{:}s.\mathsf{W}_{\mathbf{f}}(\phi_0)$ when ϕ is $\forall x{:}s.\phi_0$;
- $\exists x{:}s.\mathsf{W}_{\mathbf{f}}(\phi_0)$ when ϕ is $\exists x{:}s.\phi_0$;
- true when ϕ is true;
- false when ϕ is false;
- if $\mathsf{W}_{\mathbf{f}}(\phi_0)$ then $\mathsf{W}_{\mathbf{f}}(\phi_1)$ else $\mathsf{W}_{\mathbf{f}}(\phi_2)$ when ϕ is $\mathsf{if}_{\mathsf{bool}}\ \phi_0\ \mathsf{then}\ \phi_1\ \mathsf{else}\ \phi_2$;
- $p(\mathsf{W}'_{\mathbf{t}}(t_1), \ldots, \mathsf{W}'_{\mathbf{t}}(t_n))$ when ϕ is $(p\ t_1 \ldots t_n)$, $p \in \Sigma_P$;
- $\phi =_{\mathsf{bool}} \mathsf{True}$ when $\phi \in \mathcal{X}$;
- $f(\mathsf{W}'_{\mathbf{t}}(t_1), \ldots, \mathsf{W}'_{\mathbf{t}}(t_n)) =_{\mathsf{bool}} \mathsf{True}$ when $f \in \Sigma_F$, and ϕ is $(f\ t_1 \ldots t_n)$.

2) *Transformation* $\mathsf{W}_{\mathbf{t}}$. Next, for any SQUIRREL term t of type other than bool, the WHY3 term $\mathsf{W}_{\mathbf{t}}(t)$ is defined as follows:

- t when $t \in \mathcal{X}$;
- if $\mathsf{W_f}(t_0)$ then $\mathsf{W_t}(t_1)$ else $\mathsf{W_t}(t_2)$ when $t = \mathsf{if}_s\ t_0\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2$;
- $f(\mathsf{W'_t}(t_1), \ldots, \mathsf{W'_t}(t_n))$ when $t = (f\ t_1 \ldots t_n)$, $f \in \Sigma_F$.

3) *Transformation* $\mathsf{W'_t}$. The WHY3 term $\mathsf{W'_t}(t)$ is defined as follows:

- if $\mathsf{W_f}(t)$ then True else False when t is of type bool;
- $\mathsf{W_t}(t)$ otherwise.

Example 6: We continue our running example, and we consider the following SQUIRREL term ϕ of type bool corresponding to axiom `a1` of Section II :

$$\forall x{:}\mathsf{nat}.\big(\mathsf{inf}\,x\,(\mathsf{succ}\,x)\big).$$

Assuming that $\mathsf{inf} \in \Sigma_P$, $\mathsf{W_f}(\phi)$ is the following formula:

$$\forall x{:}\mathsf{nat}.\mathsf{inf}(x, \mathsf{succ}(x)).$$

However, in case $\mathsf{inf} \in \Sigma_F$, we have that:

$$\forall x{:}\mathsf{nat}.\big(\mathsf{inf}(x, \mathsf{succ}(x)) =_{\mathsf{bool}} \mathsf{True}\big).$$

B. Soundness

Given a SQUIRREL model $\mathcal{M}$ wrt. $(\mathcal{B}, \mathcal{F})$ and $\mathcal{X}$, and arbitrary values of $\eta \in \mathbb{N}$ and $\rho \in \mathbb{T}_{\mathcal{M},\eta}$, we define the WHY3 interpretation $\mathcal{I}(\mathcal{M}, \eta, \rho)$ wrt. $(\Sigma_S, \Sigma_F, \Sigma_P)$ and $\mathcal{X}$, as follows:

$[\![s]\!]^{\mathsf{W}}_{\mathcal{I}(\mathcal{M},\eta,\rho)} = [\![s]\!]^{\mathsf{S}}_{\mathcal{M},\eta}$ for any $s \in \mathcal{B}$;
$[\![f]\!]^{\mathsf{W}}_{\mathcal{I}(\mathcal{M},\eta,\rho)} = [\![f]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta, \rho)$ for any $f \in \Sigma_F \setminus \{\mathsf{True}, \mathsf{False}\}$;
$[\![p]\!]^{\mathsf{W}}_{\mathcal{I}(\mathcal{M},\eta,\rho)} = [\![p]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta, \rho)$ for any $p \in \Sigma_P \setminus \{=_s \mid s \in \Sigma_S\}$;
$[\![x]\!]^{\mathsf{W}}_{\mathcal{I}(\mathcal{M},\eta,\rho)} = [\![x]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta, \rho)$ for any variable $x \in \mathcal{X}$.

We now state our key lemma whose proof is available in Appendix B.

Lemma 1: Let t be a well-formed term. For any model $\mathcal{M}$, η and ρ, and $\mathcal{I}' = \mathcal{I}(\mathcal{M}, \eta, \rho)$, we have:

1) $[\![\mathsf{W_f}(t)]\!]^{\mathsf{W}}_{\mathcal{I}'} = [\![t]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta, \rho)$ when t is of type bool, and $[\![\mathsf{W_t}(t)]\!]^{\mathsf{W}}_{\mathcal{I}'} = [\![t]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta, \rho)$ otherwise;
2) $[\![\mathsf{W'_t}(t)]\!]^{\mathsf{W}}_{\mathcal{I}'} = [\![t]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta, \rho)$.

This allows us to obtain a first soundness result for our translation:

Theorem 1: Let ϕ be a well-formed SQUIRREL term of type bool. We have that: $\models^{\mathsf{W}} \mathsf{W_f}(\phi)$ implies $\models^{\mathsf{S}} \phi$.

Proof. We prove the contrapositive. We assume that $\not\models^{\mathsf{S}}\phi$, i.e. there exists $\mathcal{M}$ such that $\mathcal{M} \not\models^{\mathsf{S}}\phi$, i.e. it is not the case that ϕ is true with overwhelming probability. In particular, this means that there exists a pair (η, ρ) such that $[\![\phi]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta, \rho) = \bot$. By Lemma 1, we have that:

$$[\![\phi]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta, \rho) = [\![\mathsf{W_f}(\phi)]\!]^{\mathsf{W}}_{\mathcal{I}(\mathcal{M},\eta,\rho)}.$$

Therefore, $\mathcal{I}(\mathcal{M}, \eta, \rho)$ does not satisfy $\mathsf{W_f}(\phi)$, hence we have that $\not\models^{\mathsf{W}} \mathsf{W_f}(\phi)$. □

This first soundness result is not sufficient for our needs, as we would like our translation to be sound even when considering a class of models restricted by an axiomatic theory. We will, therefore, extend the result to $\mathcal{T}$-validity.

Theorem 2: Let ϕ be a SQUIRREL term of type bool, and $\mathcal{T}$ be a finite set of SQUIRREL terms of type bool, with all these formulas being well-formed. We have that:

$$\models^{\mathsf{W}}_{\mathsf{W_f}(\mathcal{T})} \mathsf{W_f}(\phi) \ \text{ implies } \ \models^{\mathsf{S}}_{\mathcal{T}} \phi.$$

Proof. By definition of $\models^{\mathsf{W}}_{\mathsf{W_f}(\mathcal{T})}$ and $\mathsf{W_f}$, and using Theorem 1, we have that:

$$\begin{aligned} \models^{\mathsf{W}}_{\mathsf{W_f}(\mathcal{T})} \mathsf{W_f}(\phi) \quad &\Rightarrow \quad \models^{\mathsf{W}} \big(\textstyle\bigwedge_{\psi\in\mathcal{T}} \mathsf{W_f}(\psi)\big) \Rightarrow \mathsf{W_f}(\phi) \\ &\Rightarrow \quad \models^{\mathsf{W}} \mathsf{W_f}((\textstyle\bigwedge_{\psi\in\mathcal{T}} \psi) \Rightarrow \phi) \\ &\Rightarrow \quad \models^{\mathsf{S}} \big(\textstyle\bigwedge_{\psi\in\mathcal{T}} \psi\big) \Rightarrow \phi \end{aligned}$$

As noted in Section III-C, this implies $\models^{\mathsf{S}}_{\mathcal{T}} \phi$. □

C. Applications

We seek to verify that some formula ϕ is $\mathcal{T}_S$-valid for some theory $\mathcal{T}_S$, partly described in Section III-D, which includes builtin SQUIRREL assumptions as well as protocol-specific ones. Thanks to Theorem 2, we can establish the $\mathcal{T}_S$-validity of ϕ from $\models^{\mathsf{W}}_{\mathsf{W_f}(\mathcal{T})} \mathsf{W_f}(\phi)$, for any finite $\mathcal{T} \subseteq \mathcal{T}_S$. The theory $\mathcal{T}_S$ is generally infinite, but only because cryptographic reasoning is based on axiom schemes (which represent infinite sets of axioms) involving conditions on subterms [13]. The limitation to a finite subset $\mathcal{T} \subseteq \mathcal{T}_S$ above explains why we cannot take cryptographic reasoning into account in our translation; that condition has no further implications. We review next the parts of $\mathcal{T}_S$ that we do incorporate in $\mathcal{T}$ for our implementation.

1) Axioms regarding timestamps: We first include the protocol-independent axioms given in Fig. 2. We also consider axioms on timestamps that depend on the protocol under study. This includes axioms stating that: every timestamp that happens is an action or init, as shown in Eq. (1); two distinct actions that happen produce two different timestamps, as shown in Eq. (2) and Eq. (3) in Section III-D.

We also include *dependency* and *mutual exclusion* axioms. Dependency axioms are used when modeling security protocols with a role featuring several inputs and outputs, to express that these actions have to be executed in the right order. Mutual exclusion axioms are generated when considering e.g. protocols involving conditional branching. They are used to express that the two actions $\mathsf{A_{then}}$ and $\mathsf{A_{else}}$ modeling the two branches of the conditional cannot both happen (for the same indices). Importantly, existing automated reasoning in SQUIRREL (in particular, the `auto` tactic) does not exploit dependency and mutual exclusion axioms except in very weak ways. As a result, the user often has to explicitly invoke these axioms to prove subgoals that rely on them.

2) Axioms regarding macros: In addition to the axioms given in Fig. 2, our translation also includes other axioms that specify the meaning of each macro symbol. The meaning of exec, input, and frame is protocol-independent, and fully described below:

1) $(\mathsf{cond}\ \mathsf{init})$;
2) $(\mathsf{cond}\ t)\ \Rightarrow\ (\mathsf{happens}\ t)$;

3) $\neg(\mathsf{happens}\ t) \vee t = \mathsf{init}\ \Rightarrow\ (\mathsf{input}\ t) = \mathsf{empty}$;
4) $(\mathsf{happens}\, t) \wedge t \neq \mathsf{init}\ \Rightarrow$
$(\mathsf{input}\, t) = (\mathsf{att}\,(\mathsf{frame}\,(\mathsf{pred}\, t)))$;

5) $\neg(\mathsf{happens}\ t) \vee t = \mathsf{init}\ \Rightarrow\ (\mathsf{frame}\ t) = \mathsf{empty}$;
6) $(\mathsf{happens}\, t) \wedge t \neq \mathsf{init}\ \Rightarrow$
$(\mathsf{frame}\ t) = \langle(\mathsf{exec}\ t),$
$\langle(\mathsf{if}\ (\mathsf{exec}\ t)\ \mathsf{then}\ (\mathsf{output}\ t)\ \mathsf{else}\ \mathsf{empty}),$
$(\mathsf{frame}\ (\mathsf{pred}\ t))\rangle\rangle$

7) $(\mathsf{exec}\ t)\ \Leftrightarrow\ t = \mathsf{init}\ \vee\ \big((\mathsf{exec}\ (\mathsf{pred}\ t)) \wedge (\mathsf{cond}\ t)\big)$.

We then generate, from the internal description of the protocol under consideration, axioms specifying the protocol-specific macros output, cond, and macros modeling memory cells. For example, on our running example, we generate the axioms shown in Example 4.

3) Axioms for names: SQUIRREL uses special constants called names to model random samplings performed by the protocol. Those names are indexed, which allows to model unbounded collections of random samplings: each name symbol n is declared with a type of the form $s \to s'$. When sort s' is explicitly declared as being *large*, or when $s' = \mathsf{msg}$, it is assumed that the random samplings corresponding to two distinct names in s' have a negligible probability of being equal. We incorporate this assumption in our translation. First, assuming that $\mathsf{n} : s_1 \to s'$ and $\mathsf{m} : s_2 \to s'$ are two distinct name symbols used in the protocol under study, with s' tagged "large", we generate:

$$\forall i : s_1.\forall j : s_2.\ \neg(\mathsf{n}\ i = \mathsf{m}\ j).$$

We also specify that two different instances of the same name over a large sort have a negligible probability of colliding:

$$\forall i : s_1.\forall j : s_1.\ \mathsf{n}\ i = \mathsf{n}\ j\ \Rightarrow\ i = j.$$

4) Axioms for correctness of primitives: When a built-in cryptographic primitive is declared in a SQUIRREL model, an axiom is generated to state its correctness. For example, for symmetric encryption, we will consider the following axiom:

$$\forall m, r, k.\ (\mathsf{sdec}\ (\mathsf{senc}\ m\ r\ k)\ k) = m.$$

5) Others: Every user-defined axiom can be marked as an "SMT hint", provided that it is system-independent, in which case it will be incorporated into our translated theory. Going back to our running example, the three axioms modeling the fact that inf is an order relation will be added to the theory.

VI. TRANSLATING TIMESTAMPS TO INTEGERS

We now consider an optimization of our translation, which consists in using integers to represent timestamps, and will be sound only wrt. the WHY3 interpretations obtained from SQUIRREL models representing traces as described in Section III-D. This result is presented as a new translation within the WHY3 logic, which conceptually comes after the core translation from SQUIRREL to WHY3 covered in Section V.

We fix a set $\mathcal{X}$ of variables and a WHY3 signature $(\Sigma_S, \Sigma_F, \Sigma_P)$ such that $\mathsf{timestamp} \in \Sigma_S$, and

- $\mathsf{happens}(\mathsf{timestamp}), \preceq\,(\mathsf{timestamp}, \mathsf{timestamp}) \in \Sigma_P$;
- $\mathsf{init} : \mathsf{timestamp}, \mathsf{pred}(\mathsf{timestamp}) : \mathsf{timestamp} \in \Sigma_F$.

We say that an interpretation $\mathcal{I}$ over $(\Sigma_S, \Sigma_F, \Sigma_P)$ is *canonical* when $[\![\mathsf{timestamp}]\!]^{\mathsf{w}}_{\mathcal{I}}$ is finite and $\mathcal{I}$ satisfies all the axioms listed in Fig. 2.

For any SQUIRREL model $\mathcal{M}$, and any η and ρ, the interpretation $\mathcal{I}(\mathcal{M}, \eta, \rho)$ is canonical provided that $\mathcal{M}$ belongs to the standard class of models considered in the tool for modeling protocols, i.e., as described in Section III-D. Our goal in this section is to leverage this observation, which is unused in the previous section. Note that the relevant conditions on SQUIRREL models can be axiomatized in the higher-order CCSA logic: besides axioms in Fig. 2 mentioned above, we can express in higher-order logic the finiteness of the timestamp type. Any SQUIRREL theory containing all these axioms will be called *standard*.

A. Translation

We first translate sorts, by replacing timestamp by int:

$$\mathsf{O_s}(s) = \begin{cases} \mathsf{int} \text{ when } s = \mathsf{timestamp} \\ s \text{ otherwise} \end{cases}$$

This translation targets the signature $(\Sigma'_S, \Sigma'_F, \Sigma'_P)$ defined by:

- $\Sigma'_S = \Sigma_S \setminus \{\mathsf{timestamp}\}$;
- $\Sigma'_F = \{\mathsf{max_t} : \mathsf{int}\} \cup$
$\{f(\mathsf{O_s}(s_1), \ldots, \mathsf{O_s}(s_n)) : \mathsf{O_s}(s) \mid f(s_1, \ldots, s_n) : s \in \Sigma_F\}$;
- $\Sigma'_P = \{\sim(\mathsf{int}, \mathsf{int})\} \cup$
$\{p(\mathsf{O_s}(s_1), \ldots, \mathsf{O_s}(s_n)) \mid p(s_1, \ldots, s_n) \in \Sigma_P^-\}$
where $\Sigma_P^- = \Sigma_P \setminus \{=_{\mathsf{timestamp}}\}$.

The target set of variables $\mathcal{X}'$ is the same as $\mathcal{X}$, except that variables typed timestamp in $\mathcal{X}$ are typed int in $\mathcal{X}'$.

We now define the translations of formulas and terms, noted $\mathsf{O_f}(\phi)$ and $\mathsf{O_t}(t)$ respectively. These translations are mostly straightforward: we replace timestamp by int, and use the predicate $\sim$ to translate equalities between timestamps.

1) *Transformation* $\mathsf{O_f}$. For a formula ϕ over $\mathcal{X}$ and $(\Sigma_S, \Sigma_F, \Sigma_P)$, we define the formula $\mathsf{O_f}(\phi)$ over $\mathcal{X}'$ and $(\Sigma'_S, \Sigma'_F, \Sigma'_P)$ as:

- $\mathsf{O_t}(t_1) \sim \mathsf{O_t}(t_2)$ when $\phi = (t_1 =_{\mathsf{timestamp}} t_2)$;
- $p(\mathsf{O_t}(t_1), \ldots, \mathsf{O_t}(t_n))$ when $\phi = p(t_1 \ldots t_n)$, $p \in \Sigma_P^-$;
- $\mathsf{O_f}(\phi_1) \diamond \mathsf{O_f}(\phi_2)$ when $\phi = \phi_1 \diamond \phi_2$, $\diamond \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$;
- $\neg\mathsf{O_f}(\phi_0)$when $\phi = \neg\phi_0$;
- $\forall x{:}\mathsf{O_s}(s).\mathsf{O_f}(\phi_0)$ when $\phi = \forall x{:}s.\phi_0$;

- $\exists x{:}\mathsf{O_s}(s).\mathsf{O_f}(\phi_0)$ when $\phi = \exists x{:}s.\phi_0$;
- $\mathsf{if}\ \mathsf{O_f}(\phi_0)\ \mathsf{then}\ \mathsf{O_f}(\phi_1)\ \mathsf{else}\ \mathsf{O_f}(\phi_2)$ when $\phi = \mathsf{if}\ \phi_0\ \mathsf{then}\ \phi_1\ \mathsf{else}\ \phi_2$;
- ϕ when $\phi \in \{\mathsf{true}, \mathsf{false}\}$.

2) *Transformation* $\mathsf{O_t}$. For a term t over $\mathcal{X}$ and $(\Sigma_S, \Sigma_F, \Sigma_P)$ of sort s, we define the term $\mathsf{O_t}(t)$ over $\mathcal{X}'$ and $(\Sigma'_S, \Sigma'_F, \Sigma'_P)$ of sort $\mathsf{O_s}(s)$ as:

- t when $t \in \mathcal{X}$;
- $f(\mathsf{O_t}(t_1), \ldots, \mathsf{O_t}(t_n))$ when $t = f(t_1 \ldots t_n)$, $f \in \Sigma_F$;
- $\mathsf{if}\ \mathsf{O_f}(\phi)\ \mathsf{then}\ \mathsf{O_t}(t_1)\ \mathsf{else}\ \mathsf{O_t}(t_2)$ when $t = \mathsf{if}\ \phi\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2$.

B. Soundness

In order to justify the soundness of our translation, we define a transformation on canonical interpretations.

Given a canonical interpretation $\mathcal{I}$ over $(\Sigma_S, \Sigma_F, \Sigma_P)$, we shall define its translation $\overline{\mathcal{I}}$ over $(\Sigma'_S, \Sigma'_F, \Sigma'_P)$. This interpretation interprets sorts in the same way as $\mathcal{I}$: for all $s \in \Sigma'_S$, we have that $[\![s]\!]^{\mathsf{W}}_{\mathcal{I}} = [\![s]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}}$.

To define the interpretation of functions and predicates in $\overline{\mathcal{I}}$, we introduce, for each sort $s \in \Sigma_S$, two auxiliary functions:

$$\sigma_s : [\![s]\!]^{\mathsf{W}}_{\mathcal{I}} \to [\![\mathsf{O_s}(s)]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}} \qquad \overline{\sigma}_s : [\![\mathsf{O_s}(s)]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}} \to [\![s]\!]^{\mathsf{W}}_{\mathcal{I}}$$

For $s \neq \mathsf{timestamp}$, they are defined by $\sigma_s(a) = a$ and $\overline{\sigma}_s(a) = a$. Then we set:

$$\sigma_{\mathsf{timestamp}}(a) \stackrel{\mathsf{def}}{=} \#\{a' \in [\![\mathsf{timestamp}]\!]^{\mathsf{W}}_{\mathcal{I}} \mid [\![\preceq]\!]^{\mathsf{W}}_{\mathcal{I}}(a', a)\}.$$

Finally, let undef be the unique element of $[\![\mathsf{timestamp}]\!]^{\mathsf{W}}_{\mathcal{I}}$ such that $[\![\mathsf{happens}]\!]^{\mathsf{W}}_{\mathcal{I}}(\mathsf{undef}) = \bot$. Note that this element is mapped to 0 by $\sigma_{\mathsf{timestamp}}$ as expected. We define $\overline{\sigma}_{\mathsf{timestamp}}(n_0)$ as the unique element $a \in [\![\mathsf{timestamp}]\!]^{\mathsf{W}}_{\mathcal{I}}$ such that

$$\#\{a' \in [\![\mathsf{timestamp}]\!]^{\mathsf{W}}_{\mathcal{I}} \mid [\![\preceq]\!]^{\mathsf{W}}_{\mathcal{I}}(a', a)\} = n_0$$

when it exists, and $\overline{\sigma}_{\mathsf{timestamp}}(n_0) = \mathsf{undef}$ otherwise.

As an immediate consequence of these definitions, we obtain the following lemma.

Lemma 2: For any element $a \in [\![s]\!]^{\mathsf{W}}_{\mathcal{I}}$, $\overline{\sigma}_s(\sigma_s(a)) = a$.

We now define the interpretation of special symbols in $\overline{\mathcal{I}}$:

- $[\![\mathsf{max_t}]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}} = \#\{a \in [\![\mathsf{timestamp}]\!]^{\mathsf{W}}_{\mathcal{I}} \mid [\![\mathsf{happens}]\!]^{\mathsf{W}}_{\mathcal{I}}(a)\}$;
- $[\![\mathsf{init}]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}} = 1 = \sigma_{\mathsf{timestamp}}([\![\mathsf{init}]\!]^{\mathsf{W}}_{\mathcal{I}})$;
- $[\![\mathsf{happens}]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}}(a) = \top$ if, and only if, $1 \leq a \leq [\![\mathsf{max_t}]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}}$;
- $[\![\sim]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}}(a_1, a_2)$ if, and only if, $a_1 = a_2$ or for both $i \in \{1, 2\}$, we have that $a_i \notin \{1, \ldots, [\![\mathsf{max_t}]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}}\}$;
- $[\![\preceq]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}}(a_1, a_2)$ if, and only if, $a_1 \leq a_2$ and for both $i \in \{1, 2\}$, we have that $a_i \in \{1, \ldots, [\![\mathsf{max_t}]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}}\}$;
- $[\![\mathsf{pred}]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}}(a) = \begin{cases} a - 1 & \text{when } a \in \{2, \ldots, [\![\mathsf{max_t}]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}}\} \\ 0 & \text{otherwise.} \end{cases}$

For other symbols, we set:

$$[\![x]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}} \stackrel{\mathsf{def}}{=} \sigma_s([\![x]\!]^{\mathsf{W}}_{\mathcal{I}}) \text{ for any variable } x \in \mathcal{X} \text{ of sort } s$$

$$[\![f]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}} \stackrel{\mathsf{def}}{=} a_1, ..., a_n \mapsto \sigma_s([\![f]\!]^{\mathsf{W}}_{\mathcal{I}}(\overline{\sigma}_{s_1}(a_1), \ldots, \overline{\sigma}_{s_n}(a_n))) \text{ for any } f(s_1, \ldots, s_n) : s \in \Sigma_F \setminus \{\mathsf{pred}, \mathsf{init}\}$$

$$[\![p]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}} \stackrel{\mathsf{def}}{=} a_1, ..., a_n \mapsto ([\![p]\!]^{\mathsf{W}}_{\mathcal{I}}(\overline{\sigma}_{s_1}(a_1), \ldots, \overline{\sigma}_{s_n}(a_n))) \text{ for any } p(s_1, \ldots, s_n) \in \Sigma^-_P \setminus \{\mathsf{happens}, \preceq\}$$

We may note that the interpretation $\overline{\mathcal{I}}$ as defined above satisfies the requirements given in Section IV-B for being a WHY3 interpretation. For instance, following the definitions above, we have that $[\![\mathsf{True}]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}} = [\![\mathsf{True}]\!]^{\mathsf{W}}_{\mathcal{I}} = \top$, and the equality predicates $=_s$ for $s \in \Sigma'_S$ are indeed interpreted as equality.

Before proving our main lemma to establish the soundness of our optimization, we first prove the following result. The aim is to show that the precise integer used to represent a timestamp that does not happen is not relevant when interpreting a term or a formula.

Lemma 3: Let $\mathcal{I}$ be a canonical interpretation and $x \in \mathcal{X}$ of sort $\mathsf{timestamp}$, such that $\mathcal{I} \models^{\mathsf{W}} \neg\mathsf{happens}(x)$. Let $n_0 \in \mathbb{Z}$ such that $n_0 \notin \{1, \ldots, [\![\mathsf{max_t}]\!]_{\overline{\mathcal{I}}}\}$, and $\mathcal{I}_0 = \overline{\mathcal{I}}[x \mapsto n_0]$.

1) $[\![\mathsf{O_f}(\phi)]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}} = [\![\mathsf{O_f}(\phi)]\!]^{\mathsf{W}}_{\mathcal{I}_0}$ for any formula ϕ built on $\mathcal{X}$ and $(\Sigma_S, \Sigma_F, \Sigma_P)$;
2) $\overline{\sigma}_s([\![\mathsf{O_t}(t)]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}}) = \overline{\sigma}_s([\![\mathsf{O_t}(t)]\!]^{\mathsf{W}}_{\mathcal{I}_0})$ for any term t of sort s built on $\mathcal{X}$ and $(\Sigma_S, \Sigma_F, \Sigma_P)$.

Proof. A detailed proof is provided in Appendix C, and we only provide a proof sketch here. By definition of $\overline{\sigma}_{\mathsf{timestamp}}$, we note that given $a, a_0 \in \mathbb{Z}$, we have:

$$\overline{\sigma}_{\mathsf{timestamp}}(a) = \overline{\sigma}_{\mathsf{timestamp}}(a_0) \text{ if, and only if}$$

$$a = a_0 \text{ or both } a, a_0 \notin \{1, \ldots, [\![\mathsf{max_t}]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}}\}$$

Thus, for any $a, a_0, b, b_0 \in \mathbb{Z}$ such that $\overline{\sigma}_{\mathsf{timestamp}}(a) = \overline{\sigma}_{\mathsf{timestamp}}(a_0)$, and $\overline{\sigma}_{\mathsf{timestamp}}(b) = \overline{\sigma}_{\mathsf{timestamp}}(b_0)$, we have:

$$[\![\mathsf{happens}]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}}(a) = [\![\mathsf{happens}]\!]^{\mathsf{W}}_{\mathcal{I}_0}(a_0)$$

$$[\![\sim]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}}(a, b) = [\![\sim]\!]^{\mathsf{W}}_{\mathcal{I}_0}(a_0, b_0)$$

Similar equalities also holds for pred and $\preceq$. Then, we prove the result by induction on ϕ and t. □

We now state our main soundness lemma. A detailed proof is provided in Appendix C, and we only provide a proof sketch here.

Lemma 4: For any canonical interpretation $\mathcal{I}$, we have:

1) $[\![\mathsf{O_f}(\phi)]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}} = [\![\phi]\!]^{\mathsf{W}}_{\mathcal{I}}$ for any formula ϕ built on $\mathcal{X}$ and $(\Sigma_S, \Sigma_F, \Sigma_P)$;
2) $[\![\mathsf{O_t}(t)]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}} = \sigma_s([\![t]\!]^{\mathsf{W}}_{\mathcal{I}})$ for any term t of sort s built on $\mathcal{X}$ $(\Sigma_S, \Sigma_F, \Sigma_P)$.

Proof. (Sketch) The result is proved by induction on ϕ and t. The case of quantifiers relies on Lemma 3 to generalize a quantification on a finite subset of integers to $\mathbb{Z}$.

Theorem 3: Let $\mathcal{T}_S$ be a standard SQUIRREL theory, containing a finite subset $\mathcal{T} \subseteq \mathcal{T}_S$ of well-formed formulas.

$$\mathsf{init} = 1$$
$$\mathsf{happens}(t) \Leftrightarrow 1 \leq t \wedge t \leq \mathsf{max_t}$$
$$t_1 \sim t_2 \Leftrightarrow \big((\mathsf{happens}(t_1) \vee \mathsf{happens}(t_2)) \Rightarrow t_1 = t_2\big)$$
$$t_1 \preceq t_2 \Leftrightarrow \big(\mathsf{happens}(t_1) \wedge \mathsf{happens}(t_2) \wedge t_1 \leq t_2\big)$$
$$\mathsf{pred}(t) = \mathsf{if}\ \mathsf{happens}(t)\ \mathsf{then}\ (t-1)\ \mathsf{else}\ 0$$

Fig. 5. WHY3 axioms defining special symbols for the optimization. Variables t, t_1, t_2 of sort timestamp are implicitly universally quantified.

Let $\mathcal{C}$ be a set of WHY3 formulas such that, for any canonical interpretation $\mathcal{I}$, we have $\overline{\mathcal{I}} \models^{\mathsf{W}} \mathcal{C}$. Let ϕ be a well-formed SQUIRREL formula. We have:

$$\models^{\mathsf{W}}_{\mathcal{C} \cup \mathsf{O_f}(\mathsf{W_f}(\mathcal{T}))} \mathsf{O_f}(\mathsf{W_f}(\phi)) \quad \Rightarrow \quad \models^{\mathsf{S}}_{\mathcal{T}_S} \phi$$

Proof. We proceed by contraposition. If $\not\models^{\mathsf{S}}_{\mathcal{T}_S} \phi$, then there is a SQUIRREL model $\mathcal{M}$ of $\mathcal{T}_S$ such that $\mathcal{M} \not\models^{\mathsf{S}} (\bigwedge_{\psi \in \mathcal{T}} \psi) \Rightarrow \phi$. Then, by Lemma 1, there exists η and ρ such that:

$$\mathcal{I}(\mathcal{M}, \eta, \rho) \not\models^{\mathsf{W}} \mathsf{W_f}\big((\bigwedge_{\psi \in \mathcal{T}} \psi) \Rightarrow \phi\big)$$

Because $\mathcal{M}$ is standard, $\mathcal{I}(\mathcal{M}, \eta, \rho)$ is canonical, and by Lemma 4 the associated interpretation $\overline{\mathcal{I}}$ is such that:

$$\overline{\mathcal{I}} \not\models^{\mathsf{W}} \mathsf{O_f}\big(\mathsf{W_f}\big((\bigwedge_{\psi \in \mathcal{T}} \psi) \Rightarrow \phi\big)\big)$$

By hypothesis we have that $\overline{\mathcal{I}} \models^{\mathsf{W}} \mathcal{C}$, and therefore we conclude that $\not\models^{\mathsf{W}}_{\mathcal{C} \cup \mathsf{O_f}(\mathsf{W_f}(\mathcal{T}))} \mathsf{O_f}(\mathsf{W_f}(\phi))$ as expected. □

C. Applications

Theorem 3 justifies using the optimized translation to prove the validity of well-formed SQUIRREL formulas wrt. standard SQUIRREL models. This applies to proving validity with the SQUIRREL tool. When using it, we simply choose a theory $\mathcal{C}$ containing axioms that define the special symbols on timestamps (encoded as integers), i.e. $\mathsf{happens}, \mathsf{pred}, \mathsf{init}, \preceq$ and $\sim$, which are obviously satisfied in any $\overline{\mathcal{I}}$. These axioms are listed in Fig. 5; note the use of WHY3 built-in symbols on integers such as 0, 1, $-$ and $\leq$. For the theory $\mathcal{T}$, we take the same one as described in Section V-C, except for the axioms specifying the properties expected from the SQUIRREL builtins $\mathsf{pred}, \mathsf{happens}, =, \preceq$, which are now consequences of $\mathcal{C}$.

VII. EXPERIMENTAL EVALUATION

In this section, we briefly present the implementation of our techniques in the SQUIRREL tool, and in particular the way unsupported terms are handled. Then, we experimentally evaluate our smt tactics in several ways.

A. Implementation

We have implemented both the core translation and the optimization as part of a new smt tactic in SQUIRREL. This is a modular addition to the tool: the tactic receives a SQUIRREL goal, with some information about name and primitive declarations, and about the protocol under study; from this it produces a WHY3 task that can then be sent to several provers, in parallel. The corresponding code weighs about 1500 lines of OCaml code, and is only compiled with SQUIRREL when the WHY3 API is available. The tactic provides options to control its behavior: the user can choose which SMT solvers to use, which translation to use, and can mark some axioms for inclusion in the theory generated by the tactic.

In our translations, SQUIRREL's function symbols returning booleans can be translated to function symbols or predicate symbols. In the implementation, we chose to translate builtin symbols to predicate symbols and user-defined ones to function symbols. This allows functions to be treated uniformly during the translation.

B. Unsupported terms

The scope of our theoretical translation is restricted compared to what can be written in the SQUIRREL prover. In particular, SQUIRREL allows polymorphic function symbols and polymorphic goals, even though these features are absent from the current theory [12]. In our implementation, we ignore these aspects and translate polymorphic terms in an opaque way using fresh function symbols. For example, suppose we have a polymorphic function $f : \alpha \to \alpha$ and we have to translate the following formula:

$$\forall x{:}\mathsf{timestamp}.\quad \big((f\ (f\ x)) = (f\ x) \wedge (f\ x) = \mathsf{init}\big) \Rightarrow (f\ (f\ x)) = \mathsf{init}.$$

We will not declare α and f in the WHY3 signature. Instead, two fresh symbols unsup_1, and unsup_2 of type

$$\mathsf{timestamp} \to \mathsf{timestamp}$$

will be introduced, and the formula will be translated as follows:

$$\forall x{:}\mathsf{timestamp}.\big(\mathsf{unsup}_1(x) = \mathsf{unsup}_2(x) \wedge \mathsf{unsup}_2(x) = \mathsf{init}\big) \Rightarrow \mathsf{unsup}_1(x) = \mathsf{init}.$$

This approach allows us to translate goals containing unsupported terms instead of giving up on the proof. Note that, in case the validity of these goals does not rely on the specific meaning of the unsupported terms, it may even happen that we succeed to prove them relying on our opaque translation. Several lemmas and protocols contain some forms of polymorphism or higher-order, making this feature crucial to make the translation usable on these case studies.

Formally, we define a transformation on terms that allows us to remove unsupported terms in a formula. We denote by $FV(t)$ the set of free variables of a term t and by $ty(t)$ its type. We denote by $T[\cdot]$ a *term context*, i.e. a term T with some distinguished subterms $\cdot$. Given a term t having the same type as $\cdot$, we denote by $T[t]$ the term obtained by replacing the occurrences of $\cdot$ by t in T. The transformation is defined as follows, and allows several occurrences of a given subterm to be replaced at once:

$$T[t] \rightsquigarrow T[\mathsf{unsup}(V)]$$

where $V \supseteq FV(t)$, and unsup is a fresh symbol of type $ty(V) \to ty(t)$.

Proposition 1: Let $T[\cdot]$ be a term context, and t be a term, both built on $(\mathcal{B}, \mathcal{F})$. Let $V \supseteq FV(t)$. For all model $\mathcal{M}$, there is a model $\mathcal{M}'$ on $(\mathcal{B}, \mathcal{F} \cup \{\mathsf{unsup}\})$ such that :

$$[\![T[t]]\!]^{\mathsf{s}}_{\mathcal{M}} = [\![T[\mathsf{unsup}(V)]]\!]^{\mathsf{s}}_{\mathcal{M}'}$$

Proof. Let $\mathcal{M}$ be a model for the signature $(\mathcal{B}, \mathcal{F})$. We define $\mathcal{M}'$ as an extension of $\mathcal{M}$ to the signature $(\mathcal{B}, \mathcal{F} \cup \{\mathsf{unsup}\})$, and we set:

$$[\![\mathsf{unsup}]\!]^{\mathsf{s}}_{\mathcal{M}'} = \vec{v} \mapsto [\![t]\!]^{\mathsf{s}}_{\mathcal{M}[V \mapsto \vec{v}]}.$$

We prove the result by induction on the term context. When $T[\cdot]$ does not contain any occurrence of $\cdot$, as $\mathcal{M}'$ is constructed as an extension of $\mathcal{M}$, we have that:

$$[\![T[\mathsf{unsup}(V)]]\!]^{\mathsf{s}}_{\mathcal{M}'} = [\![T[t]]\!]^{\mathsf{s}}_{\mathcal{M}'} = [\![T[t]]\!]^{\mathsf{s}}_{\mathcal{M}}.$$

When $T[\cdot] = \cdot$, we have that:

$$\begin{aligned}[\![T[\mathsf{unsup}(V)]]\!]^{\mathsf{s}}_{\mathcal{M}'} &= [\![\mathsf{unsup}(V)]\!]^{\mathsf{s}}_{\mathcal{M}'} \\ &= (\vec{v} \mapsto [\![t]\!]^{\mathsf{s}}_{\mathcal{M}[V \mapsto \vec{v}]})([\![V]\!]^{\mathsf{s}}_{\mathcal{M}'}) \\ &= (\vec{v} \mapsto [\![t]\!]^{\mathsf{s}}_{\mathcal{M}[V \mapsto \vec{v}]})([\![V]\!]^{\mathsf{s}}_{\mathcal{M}}) \\ &= [\![t]\!]^{\mathsf{s}}_{\mathcal{M}} = [\![T[t]]\!]^{\mathsf{s}}_{\mathcal{M}}.\end{aligned}$$

When $T[\cdot] = f(T_1[\cdot], \ldots, T_n[\cdot])$, as $[\![f]\!]^{\mathsf{s}}_{\mathcal{M}} = [\![f]\!]^{\mathsf{s}}_{\mathcal{M}'}$, we conclude relying on our induction hypothesis on $T_1[\cdot]$, ...,$T_n[\cdot]$.

When $T[\cdot] = \mathcal{Q}x : \tau.T_1[\cdot]$, relying on our induction hypothesis, we know that for all $\mathcal{M}$ there is $\mathcal{M}'$ such that

$$[\![T_1[t]]\!]^{\mathsf{s}}_{\mathcal{M}} = [\![T_1[\mathsf{unsup}(V)]]\!]^{\mathsf{s}}_{\mathcal{M}'}.$$

This applies to all models of the form $\mathcal{M}[x \mapsto a]$ with $a \in [\![\tau]\!]^{\mathsf{s}}_{\mathcal{M}}$, and allows us to conclude. □

This proposition guarantees that when a formula is not valid, then the formula obtained after our transformation is also not valid. This result can be generalized by transitivity to any number of iterations of the transformation.

In our implementation, this transformation is used whenever we encounter one of these cases: polymorphic or higher-order functions and quantifiers; partial applications of function symbols; λ-terms; the builtins try find and diff; as well as the builtin let ... in. We do not compute the exact set of free variables of a term when applying the transformation. Instead, we take the set of all the variables that are introduced in surrounding quantifiers.

C. SQUIRREL *benchmark*

We have developed a generic system to benchmark automated reasoning techniques at various points of SQUIRREL's proof process, and we present below some of the results that we have obtained in this way for our smt tactic. All experiments have been ran on a 12-core machine with 16GB of RAM, running Linux with Z3 4.12.6, CVC5 1.0.8, ALT-ERGO 2.5.4 and their variants. All prover calls are performed with a timeout of 10 seconds.

	True		False	
auto	2452		403	
	True	False	True	False
CVC5	2323	129	138	265
Z3	2309	143	142	262
All	2325	127	156	247

TABLE I
SMT COMPARED TO AUTO

We evaluate the performance of the smt tactic, both in terms of execution time and ability to prove goals. To do this, we compare it to another form of automated reasoning in SQUIRREL: the auto tactic. That tactic implements an ad-hoc procedure that searches for basic proofs. It also relies on the constraints tactic, which performs quantifier-free reasoning on timestamps and indices, and is believed to be complete for a core theory of timestamps that notably excludes dependency and conflict axioms. From a user's standpoint, auto is used to prove goals once most of the proof, and especially the cryptographic reasoning, has been done. We thus aim for smt to cover its uses.

We compare the smt and auto tactics on each call to auto and then on each automatic goal simplification performed by SQUIRREL, when considering all the files from the `examples/` directory of SQUIRREL's repository. For each call, we record the execution time and whether the tactic could prove the goal. We summarize the results in Table I and Table II. In these tables, we provide separate results for the two main provers, CVC5 and Z3, as well as aggregated results for all provers listed above; the results of the smt tactic are broken down according to the result of auto, indicating for the goals proved by auto (True) how many have been proved (True) or not (False) by smt, and similarly for the goals which auto could not prove (False).

1) Comparison on each call to auto (Table I): As expected, smt can prove most of the goals proved by auto and even 156 out of the 403 goals where auto fails. The cases where smt is not as effective are mainly polymorphic goals (103 cases out of the 127 where auto is better) or goals where some information is lost when unsupported subterms are translated opaquely by smt (e.g. polymorphic functions, diff and try-find constructs). The last line shows the interest of running multiple provers in parallel, to benefit from the combination of their relative strengths. To conclude on this benchmark, note that it is biased towards auto since this tactic is generally called in our example files when it can actually prove a goal.

2) Comparison on automatic goal simplifications (Table II): In this benchmark, both auto and smt are called on each automatic goal simplification performed by SQUIRREL, i.e. after each elementary tactic invocation. In particular, our two automated reasoning tactics are compared on all subgoals that the user sees, i.e. at each step of all proofs. The main results in this table are the number of cases where the tactics can prove the goal. Depending on the solver, between 3691 (i.e.,

	True		False	
auto	2780		3223	
	True	False	True	False
CVC5	2601	179	1090	2133
Z3	2585	195	1121	2102

TABLE II
SMT COMPARED TO AUTO AFTER EACH SIMPLIFICATION

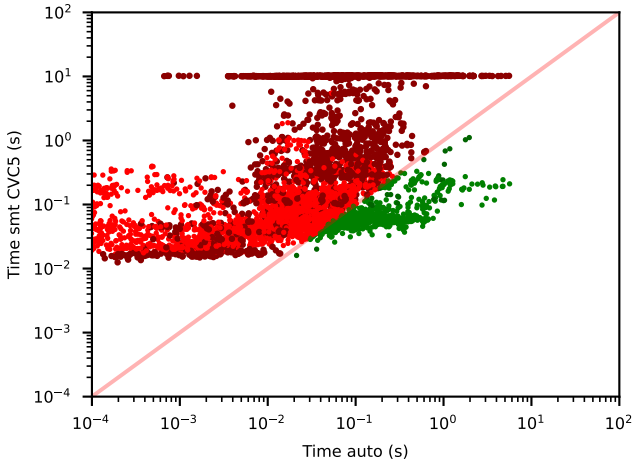

Fig. 6. Execution time smt CVC5 and auto

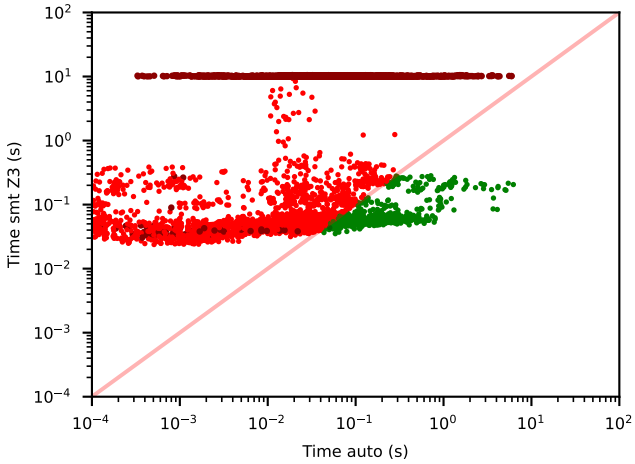

Fig. 7. Execution time smt Z3 and auto

2601 + 1090) and 3706 goals are proved by smt, compared to 2780 goals proved by auto. Notably, there are between 1090 and 1121 goals proved by smt that auto does not prove. This shows a great potential for SMT solvers to shorten, and thus facilitate, proofs — especially as these files were not built around this new tactic. There are also between 179 and 195 cases where auto can prove a goal but not smt. These cases are mainly due, once again, to the opaque translation of unsupported terms.

The results for execution times are shown respectively in Fig. 6 and Fig. 7 for CVC5 and Z3. These figures show the execution times in seconds for each call, with the time for auto on the x-axis and the time for smt on the y-axis, using a logarithmic scale in both cases. We display calls where auto is faster in red and cases where smt is faster in green. A darker tone is used when smt cannot prove the goal.

Family	auto	Core	Optimization
f_1	TO for $n > 71$	TO for $n > 38$	1.424
f_1^-	TO for $n > 71$	TO for all n	1.402
f_2	TO for $n > 81$	0.114	1.422
f_2^-	TO for $n > 82$	TO for $n > 38$	1.080
f_3	TO for $n > 1$	TO for $n > 24$	1.327

TABLE III
BEST EXECUTION TIMES (IN s) BETWEEN Z3 AND CVC5 ON FAMILIES OF FORMULAS FOR SIZE $n = 300$ (TO = TIMEOUT)

We note that the smt tactic always requires more than 10^{-2} seconds to conclude: this corresponds to the fixed cost of translating a SQUIRREL goal to a WHY3 task and off-loading it to the prover. However, smt rarely requires long execution times on success, even when auto requires higher computation times. This leads to similar average execution times between auto and smt when they succeed (0.06s and 0.08s). The execution times on failures are much higher for smt; most are caused by the 10s timeout, as displayed by the line of dark red dots on $y = 10$s. While auto requires, on average, 0.13s to conclude when it fails, smt requires 4.75s for CVC5 and 9.57s for Z3. This difference between the two solvers can be seen graphically: CVC5 stops several times before the timeout, whereas Z3 almost always reaches the timeout when it cannot prove a goal. Overall, smt remains competitive with auto if we fix a smaller timeout. Outside of this benchmark, the default timeout used for smt has been set to 1 second, which makes the tactic nicer to work with and, as shown here, does not lose much in terms of proved goals.

D. Optimization

To study the impact of our optimization on reasoning about timestamps, we evaluate both versions of our translation on five families of formulas. Families (f_1) to (f_3) are valid formulas while each (f_i^-) is an invalid variant of (f_i):

$$(\mathsf{pred}^n\ t) \preceq t' \Rightarrow ((\bigvee_{1\le k\le n} \mathsf{pred}^k\ t = t') \vee t \preceq t') \quad (f_1)$$

$$(\mathsf{pred}^n\ t) \preceq t' \Rightarrow ((\bigvee_{2\le k\le n} \mathsf{pred}^k\ t = t') \vee t \preceq t') \quad (f_1^-)$$

$$(\bigwedge_{0\le k<n} \mathsf{pred}\ t_k \preceq t_{k+1} \wedge t_{k+1} \preceq \mathsf{pred}\ t_k) \Rightarrow \mathsf{pred}^n\ t_0 = t_n \quad (f_2)$$

$$(\bigwedge_{1\le k<n} \mathsf{pred}\ t_k \preceq t_{k+1} \wedge t_{k+1} \preceq \mathsf{pred}\ t_k) \Rightarrow \mathsf{pred}^n\ t_0 = t_n \quad (f_2^-)$$

$$(t \preceq t' \wedge \mathsf{init} \preceq \mathsf{pred}^n\ t) \Rightarrow \mathsf{pred}^n\ t \preceq \mathsf{pred}^n\ t' \quad (f_3)$$

We have tested auto as well as smt, with both of our translations, on these families of functions, for increasing sizes, with a timeout of 10s. We found that auto can only prove f_1 formulas for sizes below 80 (before timeout); f_2 below 70; f_3 only for size 1. In contrast, smt can handle

	Core		Optimization	
	CVC5	Z3	CVC5	Z3
Number of True	3756	3676	3691	3706
Average time for True	0.080	0.083	0.084	0.105
Average time for False	7.814	9.533	4.757	9.567
Average time	2.975	3.747	1.884	3.726

TABLE IV
CORE TRANSLATION COMPARED TO THE OPTIMIZATION ON PROTOCOLS

formulas of size up to several hundred. We show results for $n = 300$ in Table III, where differences between our two translations start having a noticeable impact. For families (f_1) and (f_3), only the optimized version succeeds. The result obtained for family (f_2) is surprising, but may be explained by the fact that only anti-symmetry and congruence closure are needed to prove these formulas; the encoding of timestamps as integers does not help much with that. With the invalid families (f_1^-) and (f_2^-), only the optimized version concludes before the timeout.

We finally compare in Table IV the results of both translations on the benchmark presented in Section VII-C2. Depending on the solver used, there are slight differences in the number of cases where True is returned by the solver. We note a significant improvement in computation time with the optimization for CVC5. The average time for False is 3.1 seconds faster: the prover is able to stop before the timeout (leading also to a faster average time).

E. Case studies with states

In this section, we review all the protocols studied in [23]. Those protocols manipulate states, and are thus good candidates to evaluate our smt tactic. Indeed, security proofs for protocols manipulating states (e.g. counters) tend to require less cryptographic reasoning and more first-order reasoning, leading us to believe that our smt tactic would be effective.

The protocols studied are Toy Counter (our running example), CANAuth [24], YubiKey and YubiHSM [25], two variations on the OSK protocol [26], as well as two other RFID protocols [27]. We present the gains achieved with the smt tactic for each case study in Table V by comparing the number of intermediate lemmas and the number of lines of code needed to prove all the lemmas.

Overall, the smt tactic significantly simplifies the proofs for all these protocols. The smaller gain for e.g. YubiHSM and OSK-v2 comes from the fact that a large part of these files concerns equivalence properties or global formulas on which the smt tactic cannot be directly called.

As already highlighted by our running example, from a user standpoint, writing a proof script means:

1) giving the solver useful intermediate lemmas – typically those that are proved by induction or that rely on some cryptographic hypotheses – that are out of reach for smt solvers;
2) applying the relevant cryptographic hypotheses; and
3) concluding relying on the smt tactic.

	without smt tactic		with smt tactic	
	# lemmas	# LoC	# lemmas	# LoC
Toy Counter	2	15	1	3
CANAuth [24]	15	181	6	20
YubiKey [25]	4	93	1	9
YubiHSM [25]	11	256	10	145
OSK-v1 [26] (running-ex.sp)	9	190	3	24
OSK-v2 [26] (running-ex-oracle.sp)	7	178	6	108
SLK06 [27]	0	8	0	6
YPLRK05 [27]	1	52	1	12

TABLE V
PROOF SIMPLIFICATIONS WITH SMT

To illustrate the type of gain obtained by our smt tactic, we detail below a part of the security analysis performed on the CANAuth protocol.

CANAuth protocol. CANAuth is a protocol that, due to real-time constraints of CAN bus networks, avoids challenge-response mechanisms, and uses counters together with message authentication codes (hmac) to ensure freshness properties. Comparing a counter with the highest value previously received allows one to ensure that a message cannot be replayed. At each session, the sender aims to send a message msg (that we will model as a fresh name) to the receiver.

We follow the analysis performed in [24] considering the same scenario as well as the same security properties, and we detail below the security analysis to establish authentication, and the absence of replay. Informally, the protocol can be described as follows:

S : $\mathsf{ctr} := \mathsf{ctr} + 1$
output$(\langle \mathsf{ctr}, msg\rangle, \mathsf{hmac}(\langle \mathsf{ctr}, msg\rangle, sk))$

R : input(x)
if $\mathsf{hmac}(\mathsf{fst}(x), sk) = \mathsf{snd}(x)$ and $\mathsf{ctr} < \mathsf{fst}(\mathsf{fst}(x))$
then output(ok); $\mathsf{ctr} := \mathsf{fst}(\mathsf{fst}(x))$

We consider an arbitrary number of pairs of communicating parties $(\mathsf{A}\ i)$ and $(\mathsf{B}\ i)$ who can play the role of the sender and the receiver, and we assume that each pair of communicating parties has already exchanged a long-term key $(sk\ i)$ and initialized with zero their own counter $(\mathsf{cellA}\ i)$ and $(\mathsf{cellB}\ i)$ used to store the value of ctr. Note that, as each party has a memory cell, our model has an arbitrary number of memory cells.

Each party can execute the protocol many times (both as a sender and a receiver) using its pre-shared long-term key $(sk\ i)$. Thus, we will use four action identifiers denoting whether the party is A or B and whether it plays the role of the sender S or the receiver R. These actions have two parameters since a party parameterized by i can execute the protocol many times. The memory cell $(\mathsf{cellA}\ i)$ will be incremented in all the sessions launched by $(\mathsf{A}\ i)$ playing the role of the sender, and in all the sessions launched by $(\mathsf{A}\ i)$ acting as a receiver, but only when the received message has the expected format. The memory cell $(\mathsf{cellB}\ i)$ will be updated in a similar way.

The authentication property for which we will detail the security analysis can be stated informally as follows: *If A successfully received a message, then that message has indeed been previously sent by B.* Here, we consider the property where A plays the role of the receiver and B the role of the sender. The same property where the two roles are swapped can be established in a rather similar way. This property also prevents an agent A (or B) from acting both as the sender and the receiver during an exchange.

In order to establish this authentication property, we have to rely on the security of the hmac. The unforgeability of the hmac ensures that the message received by (A i) comes from one who has (sk i) in its possession. This is actually not sufficient. The proof also relies on the fact that the value of (cellA i) is increasing during the execution, and even strictly increasing at some specific steps. This actually ensures that a message successfully received by (A i) (with a current counter value larger than the one contained in the received message) comes from (B i), as (A i) can only have emitted message having smaller counter values that its current value.

The proof script to establish the authentication property relying on our smt tactic is given below.

```
lemma ctrIncA (t, t':timestamp, i:index):
  happens(t) ⇒  exec@t ⇒ t' ≺ t ⇒
  (  cellA(i)@t' ∼≺ cellA(i)@t
  || cellA(i)@t' = cellA(i)@t).
Proof. induction t. smt. Qed.

lemma authA (i,j:index) :
  happens(RA(i,j)) ⇒ exec@RA(i,j) ⇒
  (exists (j':index), SB(i,j') ≺ RA(i,j) &&
   fst(output@SB(i,j')) = fst(input@RA(i,j)))
Proof.
  intro Hap @/exec @/cond[H1 H2 H3].
  use ctrIncA.
  euf H3; smt.
Qed.
```

Listing 6. CANAuth authentication

The proof script for authA contains the two stages of reasoning we mentioned previously. Note that the lemma ctrIncA has to be explicitly used, as its proof required an induction that our smt tactic is not able to perform by itself. Apart from the call to the euf tactic, the rest of the proof can be left to smt. The first line of the proof script simply introduces hypotheses and unfolds macros: this is needed to be able to give a name to the hypothesis on which we want to apply the euf tactic.

Performing the same proof without relying on the smt tactic is cumbersome. It requires us to introduce two other intermediate lemmas to reason about states:

1) To prove ctrIncA by induction, we have to show that (cellA i) increases between two consecutive timestamps. This proof script requires a dozen of tactics, whereas this reasoning is automatically done by our smt tactic.
2) To establish authA, in addition to the lemmas already mentioned, we need to express that (cellA i) strictly increases when (A i) successfully receives a message. This lemma can then be proved by calling 8 different tactics. Relying on our smt tactic, we do not even need to explicitly state this as an intermediate lemma.

To sum up, the proof of authA without relying on our smt tactic contained more than 70 lines, whereas the entire proof script (with smt) is shown in Listing 6.

Finally, we can also express the property stating that replay attacks are impossible: the same message cannot be successfully received twice. The proof script of this property relying on our smt tactic is given below.

```
lemma noReplay (i,i',j,j':index) :
  happens(RA(i,j)) ⇒ exec@RA(i,j)
   ⇒ happens(RA(i',j')) ⇒ exec@RA(i',j')
   ⇒ (i ≠ i' || j ≠ j')
   ⇒ fst(input@RA(i,j)) ≠ fst(input@RA(i',j')).
Proof.
  use authA. use ctrIncA. smt.
Qed.
```

Listing 7. CANAuth absence of replay

Without relying on our smt tactic, the proof script for this lemma is about 20 lines of code.

VIII. Conclusion

We have shown that traditional first-order logic tools such as SMT solvers can be used to verify validity in Squirrel's local logic, despite its reliance on the cryptographers' notion of overwhelming truth. Specifically, we have designed translations from Squirrel to the Why3 logic: a first translation that applies in a very general setting, and a second one that takes advantage of a specific class of models to leverage integer-specific SMT reasoning. Both translations have been implemented in the Squirrel tool as part of a new tactic. We have extensively evaluated this tactic, showing that it effectively improves on prior automated reasoning techniques.

The simple theoretical foundations of our approach are arguments in its favor, particularly when compared to the more ambitious but much more complex CryptoVampire tool [22]. We note that our tactic has been added as a modular addition to the Squirrel system, and exploited in case studies from [23] involving protocols with states. In contrast, while the theory behind CryptoVampire seems to account for stateful protocols, the only practical successes reported in [22] are for the stateless case studies of [11]. More fundamentally, CryptoVampire builds on a variant of the meta-logic of earlier versions of Squirrel. Even if we ignore the differences in the logical setups, various limitations in the language of CryptoVampire make it difficult to meaningfully compare it with our smt tactic on our benchmark — current CryptoVampire models have been manually rewritten and adapted from early Squirrel files. In contrast with CryptoVampire, we have framed our work in a subset of Squirrel's current, higher-order CCSA logic. In addition to being more general, that setup also allows us to justify our translations without having to consider the technical but

irrelevant details of the meta-logic's trace models. Finally, our tactic is readily available (and proved sound) as part of the current system.

We are considering several directions for future work. First, we will extend our techniques to support several useful features of SQUIRREL's logic, such as higher-order quantification, polymorphism, diff and try-find operators. These new features may be handled by exploiting a richer fragment of WHY3, or by pre-processing. In either case, we expect that the simplicity of our setup will allow smooth extensions. Second, we plan to exploit more domain-specific SMT reasoning, e.g. by encoding integers manipulated in protocols as SMT integers rather than axiomatizing them. While our optimization did not fully meet our expectations, as it only provides a significant gain when the tactic fails, it can still serve as a first theoretical basis for these kinds of encodings. Third, we will explore how to make our tactic fully usable in practical SQUIRREL developments. A pre-requisite in this direction is to ensure reproducible results, e.g. by keeping track of which prover has successfully closed a goal, and which axioms have effectively been useful in that respect.

REFERENCES

[1] "SSL/TLS and PKI history." [Online]. Available: https://www.feistyduck.com/ssl-tls-and-pki-history/

[2] M. Vucinic, G. Selander, J. P. Mattsson, and T. Watteyne, "Lightweight authenticated key exchange with EDHOC," *Computer*, vol. 55, no. 4, pp. 94–100, 2022. [Online]. Available: https://doi.org/10.1109/MC.2022.3144764

[3] "Federal chancellery ordinance on electronic voting." [Online]. Available: https://www.fedlex.admin.ch/eli/cc/2022/336/en

[4] D. Dolev and A. C. Yao, "On the security of public key protocols," *IEEE Trans. Information Theory*, vol. 29, no. 2, pp. 198–207, 1983.

[5] D. A. Basin, R. Sasse, and J. Toro-Pozo, "The EMV standard: Break, fix, verify," in *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*. IEEE, 2021, pp. 1766–1781. [Online]. Available: https://doi.org/10.1109/SP40001.2021.00037

[6] T. Claverie, G. Avoine, S. Delaune, and J. Lopes-Esteves, "Tamarin-based analysis of bluetooth uncovers two practical pairing confusion attacks," in *Computer Security - ESORICS 2023 - 28th European Symposium on Research in Computer Security, The Hague, The Netherlands, September 25-29, 2023, Proceedings, Part III*, ser. Lecture Notes in Computer Science, G. Tsudik, M. Conti, K. Liang, and G. Smaragdakis, Eds., vol. 14346. Springer, 2023, pp. 100–119. [Online]. Available: https://doi.org/10.1007/978-3-031-51479-1_6

[7] S. Goldwasser and S. Micali, "Probabilistic Encryption," *J. Comput. Syst. Sci.*, vol. 28, no. 2, pp. 270–299, 1984.

[8] G. Barthe, B. Grégoire, S. Heraud, and S. Z. Béguelin, "Computer-Aided Security Proofs for the Working Cryptographer," in *Proceedings of the Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011.*, ser. Lecture Notes in Computer Science, P. Rogaway, Ed., vol. 6841. Springer, 2011, pp. 71–90.

[9] G. Barthe, B. Grégoire, and S. Zanella Béguelin, "Formal certification of code-based cryptographic proofs," in *Proceedings of the 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 2009, pp. 90–101.

[10] B. Blanchet, "A Computationally Sound Mechanized Prover for Security Protocols," in *Proceedings of the 2006 IEEE Symposium on Security and Privacy (S&P 2006), 21-24 May 2006, Berkeley, California, USA*. IEEE Computer Society, 2006, pp. 140–154.

[11] D. Baelde, S. Delaune, C. Jacomme, A. Koutsos, and S. Moreau, "An Interactive Prover for Protocol Verification in the Computational Model," in *Proceedings of the 42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*. IEEE, 2021, pp. 537–554.

[12] D. Baelde, A. Koutsos, and J. Lallemand, "A Higher-Order Indistinguishability Logic for Cryptographic Reasoning," in *LICS*, 2023, pp. 1–13.

[13] G. Bana and H. Comon-Lundh, "A Computationally Complete Symbolic Attacker for Equivalence Properties," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*. ACM, 2014, pp. 609–620.

[14] J.-C. Filliâtre, "One Logic To Use Them All," in *CADE 24 - the 24th International Conference on Automated Deduction*, M. P. Bonacina, Ed. Lake Placid, NY, United States: Springer, Jun. 2013. [Online]. Available: https://inria.hal.science/hal-00809651

[15] "Why3 – Where programs meet provers." [Online]. Available: https://www.why3.org/

[16] "Dafny." [Online]. Available: https://dafny.org/

[17] "Frama-C – Framework for Modular Analysis of C programs." [Online]. Available: https://frama-c.com/

[18] C. Abate, P. G. Haselwarter, E. Rivas, A. V. Muylder, T. Winterhalter, C. Hritcu, K. Maillard, and B. Spitters, "SSProve: A Foundational Framework for Modular Cryptographic Proofs in Coq," in *CSF*. IEEE, 2021, pp. 1–15.

[19] D. A. Basin, A. Lochbihler, and S. R. Sefidgar, "CryptHOL: Game-Based Proofs in Higher-Order Logic," *J. Cryptol.*, vol. 33, no. 2, pp. 494–566, 2020.

[20] P. Fontaine, J.-Y. Marion, S. Merz, L. P. Nieto, and A. Tiu, "Expressiveness + Automation + Soundness: Towards combining smt solvers and interactive proof assistants," in *Tools and Algorithms for the Construction and Analysis of Systems: 12th International Conference, TACAS 2006, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, Vienna, Austria, March 25-April 2, 2006. Proceedings 12*. Springer, 2006, pp. 167–181.

[21] M. Armand, G. Faure, B. Grégoire, C. Keller, L. Théry, and B. Werner, "A modular integration of sat/smt solvers to coq through proof witnesses," in *International Conference on Certified Programs and Proofs*. Springer, 2011, pp. 135–150.

[22] S. Jeanteur, L. Kovacs, M. Maffei, and M. Rawson, "CryptoVampire: Automated reasoning for the complete symbolic attacker cryptographic model," in *Proceedings of the 45th IEEE Symposium on Security and Privacy, SP 2024, San Francisco, CA, USA, 20-23 May 2024*. IEEE, 2024.

[23] D. Baelde, S. Delaune, A. Koutsos, and S. Moreau, "Cracking the Stateful Nut: Computational Proofs of Stateful Security Protocols using the Squirrel Proof Assistant," in *Proceedings of the 35th IEEE Computer Security Foundations Symposium, CSF 2022, Haifa, Israel, August 7-10, 2022*. IEEE, 2022, pp. 289–304.

[24] V. Cheval, V. Cortier, and M. Turuani, "A little more conversation, a little less action, a lot more satisfaction: Global states in proverif," in *31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, July 9-12, 2018*. IEEE Computer Society, 2018, pp. 344–358. [Online]. Available: https://doi.org/10.1109/CSF.2018.00032

[25] R. Künnemann and G. Steel, "Yubisecure? formal security analysis results for the yubikey and yubiHSM," in *Proc. 8th International Workshop on Security and Trust Management (STM'12)*, ser. Lecture Notes in Computer Science, vol. 7783. Springer, 2012, pp. 257–272.

[26] M. Ohkubo, K. Suzuki, S. Kinoshita *et al.*, "Cryptographic approach to "privacy-friendly" tags," in *RFID privacy workshop*, vol. 82. Cambridge, USA, 2003.

[27] T. van Deursen and S. Radomirovic, "Attacks on RFID protocols," *IACR Cryptol. ePrint Arch.*, vol. 2008, p. 310, 2008. [Online]. Available: http://eprint.iacr.org/2008/310

[28] G. Bana and H. Comon-Lundh, "Towards unconditional soundness: Computationally complete symbolic attacker," in *POST*, ser. Lecture Notes in Computer Science, vol. 7215. Springer, 2012, pp. 189–208.

APPENDIX A
SQUIRREL'S LOGIC

A. Relationship to the logic of [12]

Our well-formed terms can be understood as syntactic sugar over standard λ-calculus, which justifies that they form a fragment of SQUIRREL's higher-order terms. For instance, $(f\ t_1 \dots t_n)$ is just an iterated application $((f\ t_1) \dots t_n)$; $\mathsf{if}_s\ t_1\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3$ stands for $(\mathsf{ite}_s\ t_1\ t_2\ t_3)$ for some function symbol ite_s; similarly, $t_1 =_s t_2$ stands for $(=_s\ t_1\ t_2)$ and propositional constructs are just infix notations for applications of the corresponding function symbols, e.g. $t_1 \wedge t_2$ is a notation for $(\wedge\ t_1\ t_2)$. In a slightly more involved way, following the approach of Church's type theory, we also view quantifiers as syntactic sugar. For each base type s we assume a function symbol $\forall_s$, and view $\forall x : s.\ t$ as $\forall_s(\lambda x.\ t)$ — and similarly for existential quantification. This is the only (implicit) use of λ-abstractions in our well-formed terms.

We impose typing constraints on well-formed terms. To define this formally, we make use of first-order typing environments, noted $\mathcal{E}$, which are sets of typed variable declarations of the form $x : s$, where x is a variable and s is a base type. These basic environments will later be extended into more general typing environments by adding type declarations for function symbols and builtins (which are also variables when viewing well-typed terms as λ-terms). To this end, we notably view our set of typed function symbols $\mathcal{F}$ as a typing environment.

Definition 5: A well-formed term t has type $s \in \mathcal{B}$ in $\mathcal{E}$, noted $\mathcal{E} \vdash t : s$, when the corresponding λ-term has type s (according to the standard rules of simply-typed λ-calculus) in the typing environment $\mathcal{E}'$ extending $\mathcal{E}$ with $\mathcal{F}$ and the following type assignments:

$$\begin{array}{rcl} \wedge, \vee, \Rightarrow & : & \mathsf{bool} \to \mathsf{bool} \to \mathsf{bool} \\ \neg & : & \mathsf{bool} \to \mathsf{bool} \\ \forall_s, \exists_s & : & (s \to \mathsf{bool}) \to \mathsf{bool} \\ =_s & : & s \to s \to \mathsf{bool} \\ \mathsf{ite}_s & : & \mathsf{bool} \to s \to s \to s \end{array}$$

We only consider terms that are well-typed in some environment, leaving that environment implicit when it is irrelevant or obvious from the context. As a result, well-formed terms must have a base type, which means in particular that function applications must be total: for $f \in \mathcal{F}$ of type $s_1 \to \dots \to s_n \to s$, we can only form the term $(f\ t_1 \dots t_k)$ when $k = n$ and each t_i has type s_i.

B. SQUIRREL specifics regarding cryptographic aspects

Specificities related to the way names, and cryptographic primitives are modeled in SQUIRREL are recalled below.

1) Names: As it is standard in the CCSA logic (e.g. [28], [13]), SQUIRREL makes use of special constants called *names* to represent random samplings performed by honest agents.

Specifically, a SQUIRREL environment may declare any number of name symbols, each one have a type of the form $\tau_1 \to \tau_2$ where τ_2 is the domain of the name, in which values are sampled, and τ_1 is called the index type and enables the representation of an unbounded collection of samplings — in practice, the index type can be omitted when indexing is not necessary, which is equivalent to taking a singleton type unit as index type.

The semantics of names is detailed in [12]. For our purposes, it suffices to say that names model families of independent identical random samplings. For names over a domain type tagged as being *large*, it is assumed that the probability of collision of two name instances is negligible. In other words, SQUIRREL restricts to models where the following axiom scheme is overwhelmingly true, for any two distinct name symbols $\mathsf{n} : \tau_1 \to \tau_2$ and $\mathsf{m} : \tau_1' \to \tau_2$ where τ_2 is large:

$$\forall i, j.\ \mathsf{n}\ i \neq \mathsf{m}\ j \tag{5}$$

$$\forall i, j.\ \mathsf{n}\ i = \mathsf{n}\ j \Rightarrow i = j \tag{6}$$

2) Modeling honest and adversarial computations: In order to model arbitrary adversarial computations in the computational model, a special function symbol $\mathsf{att} : \mathsf{msg} \to \mathsf{msg}$ is assumed to be interpreted as a function computable by a probabilistic polynomial-time Turing machine, using the random tape ρ as the source of randomness. Most other function symbols are assumed to be computable by polynomial-time machine that are deterministic, i.e. do not use ρ.

Some function symbols implicitly come with functionality assumptions. For instance, we assume some pairing constructs over msg and restrict to models where $\mathsf{fst}\ \langle x, y \rangle = x$ is overwhelmingly true, and similarly for snd. Further, encryptions and signatures come with functionality assumptions, expressing the correctness of decryption or of signature verifications.

Function symbols declared specifically as cryptographic primitives are further constrained to be interpreted by functions that satisfy some standard cryptographic assumptions, such as PRF, EUF-CMA, etc. All these assumptions enable a specific logical treatment of cryptographic assumptions, whose details are irrelevant for the present work. As an example, if h is declared as a PRF, then $\mathsf{h}(\mathbf{true}, \mathsf{k}) \neq \mathsf{h}(\mathbf{false}, \mathsf{k})$ is valid in the considered class of models — collision-resistance is indeed a (weak) consequence of pseudo-randomness.

APPENDIX B
CORE TRANSLATION (SOUNDNESS PROOF)

Lemma 1: Let t be a well-formed term. For any model $\mathcal{M}$, η and ρ, and $\mathcal{I}' = \mathcal{I}(\mathcal{M}, \eta, \rho)$, we have:

1) $[\![\mathsf{W}_{\mathbf{f}}(t)]\!]^{\mathsf{W}}_{\mathcal{I}'} = [\![t]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta, \rho)$ when t is of type bool, and $[\![\mathsf{W}_{\mathbf{t}}(t)]\!]^{\mathsf{W}}_{\mathcal{I}'} = [\![t]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta, \rho)$ otherwise;
2) $[\![\mathsf{W}'_{\mathbf{t}}(t)]\!]^{\mathsf{W}}_{\mathcal{I}'} = [\![t]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta, \rho)$.

Proof. We prove the lemma by induction on t. We thus consider an arbitrary $\mathcal{M}, \eta, \rho$ and let $\mathcal{I}' = \mathcal{I}(\mathcal{M}, \eta, \rho)$. We first show 1):

Case $t = x$ is a variable. When x is of type bool, we have:

$$\begin{array}{rcl} [\![\mathsf{W}_{\mathbf{f}}(x)]\!]^{\mathsf{W}}_{\mathcal{I}'} & = & [\![x =_{\mathsf{bool}} \mathsf{True}]\!]^{\mathsf{W}}_{\mathcal{I}'} \\ & = & [\![x]\!]^{\mathsf{W}}_{\mathcal{I}'} = [\![x]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta, \rho). \end{array}$$

Otherwise, we have:

$$[\![\mathsf{W_t}(x)]\!]^{\mathsf{W}}_{\mathcal{I}'} = [\![x]\!]^{\mathsf{W}}_{\mathcal{I}'} = [\![x]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho).$$

Case $t = \mathsf{true}$ (resp. false). Here t is of type bool, and:

$$[\![\mathsf{W_f}(\mathsf{true})]\!]^{\mathsf{W}}_{\mathcal{I}'} = [\![\mathsf{true}]\!]^{\mathsf{W}}_{\mathcal{I}'} = \top = [\![\mathsf{true}]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho).$$

Case $t = (p\ t_1 \dots t_n)$ with $p \in \Sigma_P \setminus \{=_s \mid s \in \mathcal{B}\}$. We have that t is of type bool, and:

$$\begin{array}{ll} & [\![\mathsf{W_f}(p\ t_1 \dots t_n)]\!]^{\mathsf{W}}_{\mathcal{I}'} \\ = & [\![p(\mathsf{W'_t}(t_1),\dots,\mathsf{W'_t}(t_n))]\!]^{\mathsf{W}}_{\mathcal{I}'} \\ = & [\![p]\!]^{\mathsf{W}}_{\mathcal{I}'}([\![\mathsf{W'_t}(t_1)]\!]^{\mathsf{W}}_{\mathcal{I}'},\dots,[\![\mathsf{W'_t}(t_n)]\!]^{\mathsf{W}}_{\mathcal{I}'}) \\ = & [\![p]\!]^{\mathsf{S}}_{\mathcal{M}}([\![t_1]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho),\dots,[\![t_n]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho)) \text{ by IH} \\ = & [\![(p\ t_1 \dots t_n)]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho) = [\![t]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho) \end{array}$$

Case $t = (t_1 =_s t_2)$ with $s \in \mathcal{B}$. The reasoning is as in the previous case, because the equality predicates $=_s$ are interpreted in the same (standard) way in both $\mathcal{M}$ and $\mathcal{I}'$.

Case $t = (f\ t_1 \dots t_n)$ with $f \in \Sigma_F \setminus \{\mathsf{True},\mathsf{False}\}$. In case t is of type bool, we have that:

$$\begin{array}{ll} & [\![\mathsf{W_f}(t)]\!]^{\mathsf{W}}_{\mathcal{I}'} \\ = & [\![f(\mathsf{W'_t}(t_1),\dots,\mathsf{W'_t}(t_n)) =_{\mathsf{bool}} \mathsf{True}]\!]^{\mathsf{W}}_{\mathcal{I}'} \\ = & [\![f(\mathsf{W'_t}(t_1),\dots,\mathsf{W'_t}(t_n))]\!]^{\mathsf{W}}_{\mathcal{I}'} \\ = & [\![f]\!]^{\mathsf{W}}_{\mathcal{I}'}([\![\mathsf{W'_t}(t_1)]\!]^{\mathsf{W}}_{\mathcal{I}'},\dots,[\![\mathsf{W'_t}(t_n)]\!]^{\mathsf{W}}_{\mathcal{I}'}) \\ = & [\![f]\!]^{\mathsf{S}}_{\mathcal{M}}([\![t_1]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho),\dots,[\![t_n]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho)) \text{ by IH} \\ = & [\![(f\ t_1 \dots t_n)]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho) = [\![t]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho) \end{array}$$

Otherwise, we have that:

$$\begin{array}{ll} & [\![\mathsf{W_t}(f\ t_1 \dots t_n)]\!]^{\mathsf{W}}_{\mathcal{I}'} \\ = & [\![f(\mathsf{W'_t}(t_1),\dots,\mathsf{W'_t}(t_n))]\!]^{\mathsf{W}}_{\mathcal{I}'} \\ = & [\![(f\ t_1 \dots t_n)]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho) = [\![t]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho) \text{ as above} \end{array}$$

Case $t = \mathsf{if}_s\ t_0\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2$ for some $s \in \mathcal{B}$. In case t is of type bool, i.e. $s = \mathsf{bool}$, we have:

$$\begin{array}{ll} & [\![\mathsf{W_f}(t)]\!]^{\mathsf{W}}_{\mathcal{I}'} \\ = & [\![\mathsf{if}\ \mathsf{W_f}(t_0)\ \mathsf{then}\ \mathsf{W_f}(t_1)\ \mathsf{else}\ \mathsf{W_f}(t_2)]\!]^{\mathsf{W}}_{\mathcal{I}'} \\ = & \begin{cases} [\![\mathsf{W_f}(t_1)]\!]^{\mathsf{W}}_{\mathcal{I}'} \text{ when } [\![\mathsf{W_f}(t_0)]\!]^{\mathsf{W}}_{\mathcal{I}'} = \top \\ [\![\mathsf{W_f}(t_2)]\!]^{\mathsf{W}}_{\mathcal{I}'} \text{ otherwise} \end{cases} \\ = & \begin{cases} [\![t_1]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho) \text{ when } [\![t_0]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho) = \top \\ [\![t_2]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho) \text{ otherwise} \end{cases} \text{ by IH} \\ = & [\![\mathsf{if}_s\ t_0\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho) \\ = & [\![t]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho) \end{array}$$

When $s \neq \mathsf{bool}$, the same reasoning allows us to conclude that $[\![\mathsf{W_t}(t)]\!]^{\mathsf{W}}_{\mathcal{I}'} = [\![t]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho)$.

Case $t = \neg t'$ (resp. $t_1 \diamond t_2$ with $\diamond \in \{\wedge,\vee,\Rightarrow,\Leftrightarrow\}$.) We have that t is of type bool, and:

$$\begin{array}{rcl} [\![\mathsf{W_f}(\neg t)]\!]^{\mathsf{W}}_{\mathcal{I}'} & = & [\![\neg \mathsf{W_f}(t')]\!]^{\mathsf{W}}_{\mathcal{I}'} \\ & = & \neg [\![\mathsf{W_f}(t')]\!]^{\mathsf{W}}_{\mathcal{I}'} \\ & = & \neg [\![t']\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho) \text{ by IH} \\ & = & [\![t]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho) \end{array}$$

Case $t = \forall x{:}s.t_0$ (resp. $\exists x{:}s.t_0$) for some $s \in \mathcal{B}$. We have that t is of type bool. For $a \in [\![s]\!]^{\mathsf{W}}_{\mathcal{I}} = [\![s]\!]^{\mathsf{S}}_{\mathcal{M},\eta}$, let $\mathcal{I}'_a = \mathcal{I}'[x \mapsto a]$ and $\mathcal{M}_a = \mathcal{M}[x \mapsto \mathbb{1}_a^{\eta,\rho}]$. Note that $\mathcal{I}'_a = \mathcal{I}(\mathcal{M}_a,\eta,\rho)$. We have:

$$\begin{array}{ll} & [\![\mathsf{W_f}(t)]\!]^{\mathsf{W}}_{\mathcal{I}'} = [\![\forall x{:}s.\mathsf{W_f}(t_0)]\!]^{\mathsf{W}}_{\mathcal{I}'} \\ = & \begin{cases} \top \text{ when } [\![\mathsf{W_f}(t_0)]\!]^{\mathsf{W}}_{\mathcal{I}'_a} = \top \text{ for any } a \in [\![s]\!]^{\mathsf{W}}_{\mathcal{I}'} \\ \bot \text{ otherwise} \end{cases} \\ = & \begin{cases} \top \text{ when } [\![t_0]\!]^{\mathsf{S}}_{\mathcal{M}_a}(\eta,\rho) = \top \text{ for any } a \in [\![s]\!]^{\mathsf{S}}_{\mathcal{M},\eta} \\ \bot \text{ otherwise} \end{cases} \\ & \text{relying on our IH and } [\![s]\!]^{\mathsf{W}}_{\mathcal{I}'} = [\![s]\!]^{\mathsf{S}}_{\mathcal{M},\eta} \\ = & [\![\forall x{:}s.t_0]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho) \\ = & [\![t]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho) \end{array}$$

It remains to prove 2). It immediately follows from 1) for terms of types other than bool, because $\mathsf{W'_t}(t) = \mathsf{W_t}(t)$ by definition. For boolean terms, we have:

$$\begin{array}{rcl} [\![\mathsf{W'_t}(t)]\!]^{\mathsf{W}}_{\mathcal{I}'} & = & [\![\mathsf{if}\ \mathsf{W_f}(t)\ \mathsf{then}\ \mathsf{True}\ \mathsf{else}\ \mathsf{False}]\!]^{\mathsf{W}}_{\mathcal{I}'} \\ & = & [\![\mathsf{W_f}(t)]\!]^{\mathsf{W}} = [\![t]\!]^{\mathsf{S}}_{\mathcal{M}}(\eta,\rho) \text{ by 1)} \end{array}$$

This concludes the proof. □

Appendix C
Optimization (soundness proof)

Lemma 3: Let $\mathcal{I}$ be a canonical interpretation and $x \in \mathcal{X}$ of sort $\mathsf{timestamp}$, such that $\mathcal{I} \models^{\mathsf{W}} \neg\mathsf{happens}(x)$. Let $n_0 \in \mathbb{Z}$ such that $n_0 \notin \{1,\dots,[\![\mathsf{max_t}]\!]_{\overline{\mathcal{I}}}\}$, and $\mathcal{I}_0 = \overline{\mathcal{I}}[x \mapsto n_0]$.

1) $[\![\mathsf{O_f}(\phi)]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}} = [\![\mathsf{O_f}(\phi)]\!]^{\mathsf{W}}_{\mathcal{I}_0}$ for any formula ϕ built on $\mathcal{X}$ and $(\Sigma_S,\Sigma_F,\Sigma_P)$;
2) $\overline{\sigma}_s([\![\mathsf{O_t}(t)]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}}) = \overline{\sigma}_s([\![\mathsf{O_t}(t)]\!]^{\mathsf{W}}_{\mathcal{I}_0})$ for any term t of sort s built on $\mathcal{X}$ and $(\Sigma_S,\Sigma_F,\Sigma_P)$.

Proof. First, by definition of $\overline{\sigma}_{\mathsf{timestamp}}$, we note that given $a, a_0 \in \mathbb{Z}$, we have:

$$\overline{\sigma}_{\mathsf{timestamp}}(a) = \overline{\sigma}_{\mathsf{timestamp}}(a_0) \text{ if, and only if}$$
$$a = a_0 \text{ or } a, a_0 \notin \{1,\dots,[\![\mathsf{max_t}]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}}\}$$

Therefore, for any $a, a_0, b, b_0 \in \mathbb{Z}$ such that $\overline{\sigma}_{\mathsf{timestamp}}(a) = \overline{\sigma}_{\mathsf{timestamp}}(a_0)$, and $\overline{\sigma}_{\mathsf{timestamp}}(b) = \overline{\sigma}_{\mathsf{timestamp}}(b_0)$, we have that:

$$[\![\mathsf{happens}]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}}(a) = [\![\mathsf{happens}]\!]^{\mathsf{W}}_{\mathcal{I}_0}(a_0)$$
$$[\![\mathsf{pred}]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}}(a) = [\![\mathsf{pred}]\!]^{\mathsf{W}}_{\mathcal{I}_0}(a_0)$$
$$[\![\sim]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}}(a,b) = [\![\sim]\!]^{\mathsf{W}}_{\mathcal{I}_0}(a_0,b_0)$$
$$[\![\preceq]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}}(a,b) = [\![\preceq]\!]^{\mathsf{W}}_{\mathcal{I}_0}(a_0,b_0)$$

We then establish the two results stated in the lemma simultaneously by structural induction on ϕ and t. We thus consider an arbitrary canonical interpretation $\mathcal{I}$ and its associated interpretation $\overline{\mathcal{I}}$, as well as $\mathcal{I}_0 = \overline{\mathcal{I}}[x \mapsto n_0]$.

We first consider the cases where the transformation $\mathsf{O_f}$ is applied on a WHY3 formula, and we distinguish several cases.

Case $\phi = \mathsf{happens}(t)$. Applying our definitions, we have that:

- $[\![\mathsf{O_f}(\mathsf{happens}(t))]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}} = [\![\mathsf{happens}]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}}([\![\mathsf{O_t}(t)]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}})$, and
- $[\![\mathsf{O_f}(\mathsf{happens}(t))]\!]^{\mathsf{W}}_{\mathcal{I}_0} = [\![\mathsf{happens}]\!]^{\mathsf{W}}_{\mathcal{I}_0}([\![\mathsf{O_t}(t)]\!]^{\mathsf{W}}_{\mathcal{I}_0})$.

Relying on our induction hypothesis (item 2 on t of sort timestamp), we know that:

$$\overline{\sigma}_{\mathsf{timestamp}}(\llbracket \mathsf{O_t}(t) \rrbracket^{\mathsf{W}}_{\mathcal{I}}) = \overline{\sigma}_{\mathsf{timestamp}}(\llbracket \mathsf{O_t}(t) \rrbracket^{\mathsf{W}}_{\mathcal{I}_0}).$$

As $\llbracket \mathsf{O_t}(t) \rrbracket^{\mathsf{W}}_{\mathcal{I}}$, and $\llbracket \mathsf{O_t}(t) \rrbracket^{\mathsf{W}}_{\mathcal{I}_0}$ are both in $\mathbb{Z}$, thanks to our remark, we know that

$$\llbracket \mathsf{happens} \rrbracket^{\mathsf{W}}_{\mathcal{I}}(\llbracket \mathsf{O_t}(t) \rrbracket^{\mathsf{W}}_{\mathcal{I}}) = \llbracket \mathsf{happens} \rrbracket^{\mathsf{W}}_{\mathcal{I}_0}(\llbracket \mathsf{O_t}(t) \rrbracket^{\mathsf{W}}_{\mathcal{I}_0}).$$

This allows us to conclude.

Case $\phi = t_1 =_{\mathsf{timestamp}} t_2$ (resp. $t_1 \preceq t_2$). These two cases can be solved as the previous one.

Case $\phi = p(t_1, \ldots, t_n)$ with $p \in \Sigma_P$ but not $\preceq$, happens, $=_s$. Applying our definitions, and assuming that the arity of p is $p(s_1, \ldots, s_n)$, we have that:

$$\llbracket \mathsf{O_f}(p(t_1, \ldots, t_n)) \rrbracket^{\mathsf{W}}_{\mathcal{I}} = \llbracket p \rrbracket^{\mathsf{W}}_{\mathcal{I}}(\overline{\sigma}_{s_1}(\llbracket \mathsf{O_t}(t_1) \rrbracket^{\mathsf{W}}_{\mathcal{I}}), \ldots, \overline{\sigma}_{s_n}(\llbracket \mathsf{O_t}(t_n) \rrbracket^{\mathsf{W}}_{\mathcal{I}}))$$

We also have that:

$$\begin{aligned}
&\llbracket \mathsf{O_f}(p(t_1, \ldots, t_n)) \rrbracket^{\mathsf{W}}_{\mathcal{I}_0} \\
&= \llbracket p \rrbracket^{\mathsf{W}}_{\mathcal{I}_0}(\llbracket \mathsf{O_t}(t_1) \rrbracket^{\mathsf{W}}_{\mathcal{I}_0}, \ldots, \llbracket \mathsf{O_t}(t_n) \rrbracket^{\mathsf{W}}_{\mathcal{I}_0}) \\
&= \llbracket p \rrbracket^{\mathsf{W}}_{\mathcal{I}}(\llbracket \mathsf{O_t}(t_1) \rrbracket^{\mathsf{W}}_{\mathcal{I}_0}, \ldots, \llbracket \mathsf{O_t}(t_n) \rrbracket^{\mathsf{W}}_{\mathcal{I}_0}) \text{ as } \mathcal{I}_0 = \overline{\mathcal{I}}[x \mapsto n_0] \\
&= \llbracket p \rrbracket^{\mathsf{W}}_{\mathcal{I}}(\overline{\sigma}_{s_1}(\llbracket \mathsf{O_t}(t_1) \rrbracket^{\mathsf{W}}_{\mathcal{I}_0}), \ldots, \overline{\sigma}_{s_n}(\llbracket \mathsf{O_t}(t_n) \rrbracket^{\mathsf{W}}_{\mathcal{I}_0})) \\
&\qquad \text{by definition of } \llbracket p \rrbracket^{\mathsf{W}}_{\mathcal{I}}
\end{aligned}$$

Relying on our induction hypothesis (item 2)), we deduce that

$$\llbracket \mathsf{O_f}(p(t_1, \ldots, t_n)) \rrbracket^{\mathsf{W}}_{\mathcal{I}} = \llbracket \mathsf{O_f}(p(t_1, \ldots, t_n)) \rrbracket^{\mathsf{W}}_{\mathcal{I}_0}.$$

Case $\phi = (t_1 =_s t_2)$ with $s \in \Sigma_S \setminus \{\mathsf{timestamp}\}$. Applying our definitions, we have that:

$$\begin{aligned}
&\llbracket \mathsf{O_f}(t_1 =_s t_2) \rrbracket^{\mathsf{W}}_{\mathcal{I}} = \\
&\qquad \llbracket =_s \rrbracket^{\mathsf{W}}_{\mathcal{I}}(\llbracket \mathsf{O_t}(t_1) \rrbracket^{\mathsf{W}}_{\mathcal{I}}, \llbracket \mathsf{O_t}(t_2) \rrbracket^{\mathsf{W}}_{\mathcal{I}}) \\
&\qquad \llbracket =_s \rrbracket^{\mathsf{W}}_{\mathcal{I}}(\llbracket \mathsf{O_t}(t_1) \rrbracket^{\mathsf{W}}_{\mathcal{I}}, \llbracket \mathsf{O_t}(t_2) \rrbracket^{\mathsf{W}}_{\mathcal{I}}) \text{ as } \llbracket =_s \rrbracket^{\mathsf{W}}_{\mathcal{I}} = \llbracket =_s \rrbracket^{\mathsf{W}}_{\mathcal{I}}
\end{aligned}$$

We also have that:

$$\begin{aligned}
&\llbracket \mathsf{O_f}(t_1 =_s t_2) \rrbracket^{\mathsf{W}}_{\mathcal{I}_0} \\
&= \llbracket =_s \rrbracket^{\mathsf{W}}_{\mathcal{I}_0}(\llbracket \mathsf{O_t}(t_1) \rrbracket^{\mathsf{W}}_{\mathcal{I}_0} \llbracket \mathsf{O_t}(t_2) \rrbracket^{\mathsf{W}}_{\mathcal{I}_0}) \\
&= \llbracket =_s \rrbracket^{\mathsf{W}}_{\mathcal{I}}(\llbracket \mathsf{O_t}(t_1) \rrbracket^{\mathsf{W}}_{\mathcal{I}_0}, \llbracket \mathsf{O_t}(t_n) \rrbracket^{\mathsf{W}}_{\mathcal{I}_0}) \text{ as } \llbracket =_s \rrbracket^{\mathsf{W}}_{\mathcal{I}} = \llbracket =_s \rrbracket^{\mathsf{W}}_{\mathcal{I}}
\end{aligned}$$

As $s \neq \mathsf{timestamp}$, we have that $\overline{\sigma}_s(t) = t$ for any term t. Therefore, for $i \in \{1, 2\}$, we have that:

- $\overline{\sigma}_s(\llbracket \mathsf{O_t}(t_i) \rrbracket^{\mathsf{W}}_{\mathcal{I}}) = \llbracket \mathsf{O_t}(t_i) \rrbracket^{\mathsf{W}}_{\mathcal{I}}$; and
- $\overline{\sigma}_s(\llbracket \mathsf{O_t}(t_i) \rrbracket^{\mathsf{W}}_{\mathcal{I}_0}) = \llbracket \mathsf{O_t}(t_i) \rrbracket^{\mathsf{W}}_{\mathcal{I}_0}$

Relying on our induction hypothesis (item 2)), for $i \in \{1, 2\}$, we have that:

$$\overline{\sigma}_s(\llbracket \mathsf{O_t}(t_i) \rrbracket^{\mathsf{W}}_{\mathcal{I}}) = \overline{\sigma}_s(\llbracket \mathsf{O_t}(t_i) \rrbracket^{\mathsf{W}}_{\mathcal{I}_0})$$

Therefore, we conclude that:

$$\llbracket \mathsf{O_f}(t_1 =_s t_2) \rrbracket^{\mathsf{W}}_{\mathcal{I}} = \llbracket \mathsf{O_f}(t_1 =_s t_2) \rrbracket^{\mathsf{W}}_{\mathcal{I}_0}.$$

Case $\phi = \mathsf{if}\ \phi_0\ \mathsf{then}\ \phi_1\ \mathsf{else}\ \phi_2$. We have that:

$$\begin{aligned}
&\llbracket \mathsf{O_f}(\mathsf{if}\ \phi_0\ \mathsf{then}\ \phi_1\ \mathsf{else}\ \phi_2) \rrbracket^{\mathsf{W}}_{\mathcal{I}} \\
&\quad = \llbracket \mathsf{if}\ \mathsf{O_f}(\phi_0)\ \mathsf{then}\ \mathsf{O_f}(\phi_1)\ \mathsf{else}\ \mathsf{O_f}(\phi_2) \rrbracket^{\mathsf{W}}_{\mathcal{I}} \\
&\quad = \begin{cases} \llbracket \mathsf{O_f}(\phi_1) \rrbracket^{\mathsf{W}}_{\mathcal{I}} \text{ when } \llbracket \mathsf{O_f}(\phi_0) \rrbracket^{\mathsf{W}}_{\mathcal{I}} = \top \\ \llbracket \mathsf{O_f}(\phi_2) \rrbracket^{\mathsf{W}}_{\mathcal{I}} \text{ otherwise} \end{cases} \\
&\quad = \begin{cases} \llbracket \mathsf{O_f}(\phi_1) \rrbracket^{\mathsf{W}}_{\mathcal{I}_0} \text{ when } \llbracket \mathsf{O_f}(\phi_0) \rrbracket^{\mathsf{W}}_{\mathcal{I}_0} = \top \\ \llbracket \mathsf{O_f}(\phi_2) \rrbracket^{\mathsf{W}}_{\mathcal{I}_0} \text{ otherwise} \end{cases} \\
&\qquad \text{applying the induction hypothesis item 1} \\
&\quad = \llbracket \mathsf{if}\ \mathsf{O_f}(\phi_0)\ \mathsf{then}\ \mathsf{O_f}(\phi_1)\ \mathsf{else}\ \mathsf{O_f}(\phi_2) \rrbracket^{\mathsf{W}}_{\mathcal{I}_0} \\
&\quad = \llbracket \mathsf{O_f}(\mathsf{if}\ \phi_0\ \mathsf{then}\ \phi_1\ \mathsf{else}\ \phi_2) \rrbracket^{\mathsf{W}}_{\mathcal{I}_0} = \llbracket \mathsf{O_f}(\phi) \rrbracket^{\mathsf{W}}_{\mathcal{I}_0}
\end{aligned}$$

Case $\phi = \forall y{:}s.\phi_0$ (resp $\exists y{:}s.\phi_0$). Let $\overline{\mathcal{I}}^+ = \overline{\mathcal{I}}[y \mapsto a]$, and $\mathcal{I}_0^+ = \mathcal{I}_0[y \mapsto a]$. We have that:

$$\begin{aligned}
&\llbracket \mathsf{O_f}(\forall y{:}s.\phi_0) \rrbracket^{\mathsf{W}}_{\mathcal{I}} = \llbracket \forall y{:}s.\mathsf{O_f}(\phi_0) \rrbracket^{\mathsf{W}}_{\mathcal{I}} \\
&= \begin{cases} \top \text{ when } \llbracket \mathsf{O_f}(\phi_0) \rrbracket^{\mathsf{W}}_{\mathcal{I}^+} = \top \text{ for any } a \in \llbracket s \rrbracket^{\mathsf{W}}_{\mathcal{I}} \\ \bot \text{ otherwise} \end{cases} \\
&= \begin{cases} \top \text{ when } \llbracket \mathsf{O_f}(\phi_0) \rrbracket^{\mathsf{W}}_{\mathcal{I}_0^+} = \top \text{ for any } a \in \llbracket s \rrbracket^{\mathsf{W}}_{\mathcal{I}_0} \\ \bot \text{ otherwise} \end{cases} \\
&\text{relying on our IH and } \llbracket s \rrbracket^{\mathsf{W}}_{\mathcal{I}} = \llbracket s \rrbracket^{\mathsf{W}}_{\mathcal{I}_0} \\
&= \llbracket \forall y{:}s.\mathsf{O_f}(\phi_0) \rrbracket^{\mathsf{W}}_{\mathcal{I}_0} \\
&= \llbracket \mathsf{O_f}(\forall y{:}s.\phi_0) \rrbracket^{\mathsf{W}}_{\mathcal{I}_0}
\end{aligned}$$

Now, we consider the cases where the transformation $\mathsf{O_t}$ is applied on a WHY3 term, and we distinguish several cases. *Case t is a variable.* By definition, we have that $\mathsf{O_t}(t) = t$. In case t is of sort $s \neq \mathsf{timestamp}$, then we have that:

$$\begin{aligned}
\overline{\sigma}_s(\llbracket \mathsf{O_t}(t) \rrbracket^{\mathsf{W}}_{\mathcal{I}}) &= \overline{\sigma}_s(\llbracket t \rrbracket^{\mathsf{W}}_{\mathcal{I}}) \\
&= \overline{\sigma}_s(\llbracket t \rrbracket^{\mathsf{W}}_{\mathcal{I}_0}) \\
&= \overline{\sigma}_s(\llbracket \mathsf{O_t}(t) \rrbracket^{\mathsf{W}}_{\mathcal{I}_0})
\end{aligned}$$

Otherwise, we have that t is of sort timestamp. We first consider the case where $t = x$. We have that:

$$\begin{aligned}
\overline{\sigma}_{\mathsf{timestamp}}(\llbracket \mathsf{O_t}(x) \rrbracket^{\mathsf{W}}_{\mathcal{I}}) &= \overline{\sigma}_{\mathsf{timestamp}}(\llbracket x \rrbracket^{\mathsf{W}}_{\mathcal{I}}) \\
&= \overline{\sigma}_{\mathsf{timestamp}}(0) \\
&= \mathsf{undef} \\
&= \overline{\sigma}_{\mathsf{timestamp}}(n_0) \\
&= \overline{\sigma}_{\mathsf{timestamp}}(\llbracket x \rrbracket^{\mathsf{W}}_{\mathcal{I}_0}) \\
&= \overline{\sigma}_{\mathsf{timestamp}}(\llbracket \mathsf{O_t}(x) \rrbracket^{\mathsf{W}}_{\mathcal{I}_0})
\end{aligned}$$

Otherwise, we have that t is of sort timestamp but $t \neq x$. We have that:

$$\begin{aligned}
\overline{\sigma}_{\mathsf{timestamp}}(\llbracket \mathsf{O_t}(t) \rrbracket^{\mathsf{W}}_{\mathcal{I}}) &= \overline{\sigma}_{\mathsf{timestamp}}(\llbracket t \rrbracket^{\mathsf{W}}_{\mathcal{I}}) \\
&= \overline{\sigma}_{\mathsf{timestamp}}(\llbracket t \rrbracket^{\mathsf{W}}_{\mathcal{I}_0}) \\
&= \overline{\sigma}_{\mathsf{timestamp}}(\llbracket \mathsf{O_t}(t) \rrbracket^{\mathsf{W}}_{\mathcal{I}_0})
\end{aligned}$$

Case $t = \mathsf{pred}(t_0)$. Applying our definitions, we have that:

- $\llbracket \mathsf{O_t}(\mathsf{pred}(t_0)) \rrbracket^{\mathsf{W}}_{\mathcal{I}} = \llbracket \mathsf{pred} \rrbracket^{\mathsf{W}}_{\mathcal{I}}(\llbracket \mathsf{O_t}(t_0) \rrbracket^{\mathsf{W}}_{\mathcal{I}})$, and
- $\llbracket \mathsf{O_t}(\mathsf{pred}(t_0)) \rrbracket^{\mathsf{W}}_{\mathcal{I}_0} = \llbracket \mathsf{pred} \rrbracket^{\mathsf{W}}_{\mathcal{I}_0}(\llbracket \mathsf{O_t}(t_0) \rrbracket^{\mathsf{W}}_{\mathcal{I}_0})$.

Relying on our induction hypothesis (item 2 on t_0 of sort timestamp), we know that:

$$\overline{\sigma}_{\mathsf{timestamp}}(\llbracket \mathsf{O_t}(t_0) \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}) = \overline{\sigma}_{\mathsf{timestamp}}(\llbracket \mathsf{O_t}(t_0) \rrbracket^{\mathbb{W}}_{\mathcal{I}_0}).$$

As $\llbracket \mathsf{O_t}(t_0) \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}$, and $\llbracket \mathsf{O_t}(t_0) \rrbracket^{\mathbb{W}}_{\mathcal{I}_0}$ are both in $\mathbb{Z}$, thanks to our remark, we know that

$$\llbracket \mathsf{pred} \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}(\llbracket \mathsf{O_t}(t_0) \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}) = \llbracket \mathsf{pred} \rrbracket^{\mathbb{W}}_{\mathcal{I}_0}(\llbracket \mathsf{O_t}(t_0) \rrbracket^{\mathbb{W}}_{\mathcal{I}_0}).$$

This allows us to obtain that:

$$\llbracket \mathsf{O_t}(\mathsf{pred}(t_0)) \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}} = \llbracket \mathsf{O_t}(\mathsf{pred}(t_0)) \rrbracket^{\mathbb{W}}_{\mathcal{I}_0}.$$

Hence, we conclude for this case.

Case $t = f(t_1, \ldots, t_n)$ with $f(s_1, \ldots, s_n) : s \in \Sigma_F \setminus \{\mathsf{pred}\}$. Relying on our induction hypothesis, and on the fact that $\mathcal{I}_0 = \overline{\mathcal{I}}[x \mapsto n_0]$, we have that:

$$\begin{aligned}
&\overline{\sigma}_s(\llbracket \mathsf{O_t}(f(t_1, \ldots, t_n)) \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}) \\
&= \overline{\sigma}_s(\sigma_s(\llbracket f \rrbracket^{\mathbb{W}}_{\mathcal{I}}(\overline{\sigma}_{s_1}(\llbracket \mathsf{O_t}(t_1) \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}), \ldots, \overline{\sigma}_{s_n}(\llbracket \mathsf{O_t}(t_n) \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}})))) \\
&= \overline{\sigma}_s(\sigma_s(\llbracket f \rrbracket^{\mathbb{W}}_{\mathcal{I}}(\overline{\sigma}_{s_1}(\llbracket \mathsf{O_t}(t_1) \rrbracket^{\mathbb{W}}_{\mathcal{I}_0}), \ldots, \overline{\sigma}_{s_n}(\llbracket \mathsf{O_t}(t_n) \rrbracket^{\mathbb{W}}_{\mathcal{I}_0})))) \\
&= \overline{\sigma}_s(\llbracket f \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}(\llbracket \mathsf{O_t}(t_1) \rrbracket^{\mathbb{W}}_{\mathcal{I}_0}, \ldots, \llbracket \mathsf{O_t}(t_n) \rrbracket^{\mathbb{W}}_{\mathcal{I}_0})) \\
&= \overline{\sigma}_s(\llbracket f \rrbracket^{\mathbb{W}}_{\mathcal{I}_0}(\llbracket \mathsf{O_t}(t_1) \rrbracket^{\mathbb{W}}_{\mathcal{I}_0}, \ldots, \llbracket \mathsf{O_t}(t_n) \rrbracket^{\mathbb{W}}_{\mathcal{I}_0})) \\
&= \overline{\sigma}_s(\llbracket \mathsf{O_t}(f(t_1, \ldots, t_n)) \rrbracket^{\mathbb{W}}_{\mathcal{I}_0})
\end{aligned}$$

Case $t = \mathsf{if}\ \phi\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2$ with t_1 and t_2 of sort s. We have that:

$$\begin{aligned}
&\overline{\sigma}_s(\llbracket \mathsf{O_t}(\mathsf{if}\ \phi\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2) \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}) \\
&\quad = \overline{\sigma}_s(\llbracket \mathsf{if}\ \mathsf{O_f}(\phi)\ \mathsf{then}\ \mathsf{O_t}(t_1)\ \mathsf{else}\ \mathsf{O_t}(t_2) \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}) \\
&\quad = \begin{cases} \overline{\sigma}_s(\llbracket \mathsf{O_t}(t_1) \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}) & \text{when } \llbracket \mathsf{O_f}(\phi) \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}} = \top \\ \overline{\sigma}_s(\llbracket \mathsf{O_t}(t_2) \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}) & \text{otherwise} \end{cases} \\
&\quad = \begin{cases} \overline{\sigma}_s(\llbracket \mathsf{O_t}(t_1) \rrbracket^{\mathbb{W}}_{\mathcal{I}_0}) & \text{when } \llbracket \mathsf{O_f}(\phi) \rrbracket^{\mathbb{W}}_{\mathcal{I}_0} = \top \\ \overline{\sigma}_s(\llbracket \mathsf{O_t}(t_2) \rrbracket^{\mathbb{W}}_{\mathcal{I}_0}) & \text{otherwise} \end{cases} \\
&\qquad \text{applying the induction hypothesis items 1 \& 2} \\
&\quad = \overline{\sigma}_s(\llbracket \mathsf{if}\ \mathsf{O_f}(\phi)\ \mathsf{then}\ \mathsf{O_t}(t_1)\ \mathsf{else}\ \mathsf{O_t}(t_2) \rrbracket^{\mathbb{W}}_{\mathcal{I}_0}) \\
&\quad = \overline{\sigma}_s(\llbracket \mathsf{O_t}(\mathsf{if}\ \phi\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2) \rrbracket^{\mathbb{W}}_{\mathcal{I}_0}) \quad = \overline{\sigma}_s(\llbracket \mathsf{O_t}(t) \rrbracket^{\mathbb{W}}_{\mathcal{I}_0})
\end{aligned}$$

This concludes the proof. □

Lemma 4: For any canonical interpretation $\mathcal{I}$, we have:

1) $\llbracket \mathsf{O_f}(\phi) \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}} = \llbracket \phi \rrbracket^{\mathbb{W}}_{\mathcal{I}}$ for any formula ϕ built on $\mathcal{X}$ and $(\Sigma_S, \Sigma_F, \Sigma_P)$;
2) $\llbracket \mathsf{O_t}(t) \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}} = \sigma_s(\llbracket t \rrbracket^{\mathbb{W}}_{\mathcal{I}})$ for any term t of sort s built on $\mathcal{X}$ $(\Sigma_S, \Sigma_F, \Sigma_P)$.

Proof. We establish the two results simultaneously by induction on the size of ϕ and t.

We first consider the different cases for building a formula.

Case $\phi = \mathsf{happens}(t)$. We have that:

$$\begin{aligned}
\llbracket \mathsf{O_f}(\mathsf{happens}(t)) \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}} &= \llbracket \mathsf{happens} \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}(\llbracket \mathsf{O_t}(t) \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}) \\
&= \llbracket \mathsf{happens} \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}(\sigma_{\mathsf{timestamp}}(\llbracket t \rrbracket^{\mathbb{W}}_{\mathcal{I}})) \text{ by IH}
\end{aligned}$$

By definition, we have that $\llbracket \mathsf{happens}(t) \rrbracket^{\mathbb{W}}_{\mathcal{I}} = \top$ if, and only if, $1 \leq \sigma_{\mathsf{timestamp}}(\llbracket t \rrbracket^{\mathbb{W}}_{\mathcal{I}}) \leq \llbracket \mathsf{max_t} \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}$, and thus we conclude that:

$$\begin{aligned}
\llbracket \mathsf{O_f}(\mathsf{happens}(t)) \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}} &= \llbracket \mathsf{happens} \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}(\sigma_{\mathsf{timestamp}}(\llbracket t \rrbracket^{\mathbb{W}}_{\mathcal{I}})) \\
&= \llbracket \mathsf{happens}(t) \rrbracket^{\mathbb{W}}_{\mathcal{I}}.
\end{aligned}$$

Case $\phi = (t_1 =_{\mathsf{timestamp}} t_2)$. We have that:

$$\begin{aligned}
&\llbracket \mathsf{O_f}(t_1 =_{\mathsf{timestamp}} t_2) \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}} \\
&= \llbracket \sim \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}(\llbracket \mathsf{O_t}(t_1) \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}, \llbracket \mathsf{O_t}(t_1) \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}) \\
&= \llbracket \sim \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}(\sigma_{\mathsf{timestamp}}(\llbracket t_1 \rrbracket^{\mathbb{W}}_{\mathcal{I}}), \sigma_{\mathsf{timestamp}}(\llbracket t_2 \rrbracket^{\mathbb{W}}_{\mathcal{I}})) \text{ by IH}
\end{aligned}$$

By definition of $\sigma_{\mathsf{timestamp}}$, we have that:

$$0 \leq \sigma_{\mathsf{timestamp}}(\llbracket t_i \rrbracket^{\mathbb{W}}_{\mathcal{I}}) \leq \llbracket \mathsf{max_t} \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}} \quad \text{for } i \in \{1, 2\}.$$

Thus we have that $\sigma_{\mathsf{timestamp}}(\llbracket t_1 \rrbracket^{\mathbb{W}}_{\mathcal{I}}) = \sigma_{\mathsf{timestamp}}(\llbracket t_2 \rrbracket^{\mathbb{W}}_{\mathcal{I}})$ if, and only if, $\llbracket \sim \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}(\sigma_{\mathsf{timestamp}}(\llbracket t_1 \rrbracket^{\mathbb{W}}_{\mathcal{I}}), \sigma_{\mathsf{timestamp}}(\llbracket t_2 \rrbracket^{\mathbb{W}}_{\mathcal{I}})) = \top$. By definition, $\llbracket t_1 =_{\mathsf{timestamp}} t_2 \rrbracket^{\mathbb{W}}_{\mathcal{I}} = \top$ if, and only if,

- either $\llbracket \mathsf{happens}(t_1) \rrbracket^{\mathbb{W}}_{\mathcal{I}} = \top$ and $\sigma_{\mathsf{timestamp}}(\llbracket t_1 \rrbracket^{\mathbb{W}}_{\mathcal{I}}) = \sigma_{\mathsf{timestamp}}(\llbracket t_2 \rrbracket^{\mathbb{W}}_{\mathcal{I}})$;
- or $\llbracket \mathsf{happens}(t_1) \rrbracket^{\mathbb{W}}_{\mathcal{I}} = \bot$ and $\sigma_{\mathsf{timestamp}}(\llbracket t_1 \rrbracket^{\mathbb{W}}_{\mathcal{I}}) = \sigma_{\mathsf{timestamp}}(\llbracket t_2 \rrbracket^{\mathbb{W}}_{\mathcal{I}}) = 0$.

Therefore, we conclude that:

$$\begin{aligned}
&\llbracket \mathsf{O_f}(t_1 =_{\mathsf{timestamp}} t_2) \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}} \\
&= \llbracket \sim \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}(\sigma_{\mathsf{timestamp}}(\llbracket t_1 \rrbracket^{\mathbb{W}}_{\mathcal{I}}), \sigma_{\mathsf{timestamp}}(\llbracket t_2 \rrbracket^{\mathbb{W}}_{\mathcal{I}})) \\
&= \llbracket t_1 =_{\mathsf{timestamp}} t_2 \rrbracket^{\mathbb{W}}_{\mathcal{I}}
\end{aligned}$$

Case $\phi = (t_1 \preceq t_2)$. We have that:

$$\begin{aligned}
&\llbracket \mathsf{O_f}(t_1 \preceq t_2) \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}} \\
&= \llbracket \preceq \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}(\llbracket \mathsf{O_t}(t_1) \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}, \llbracket \mathsf{O_t}(t_1) \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}) \\
&= \llbracket \preceq \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}(\sigma_{\mathsf{timestamp}}(\llbracket t_1 \rrbracket^{\mathbb{W}}_{\mathcal{I}}), \sigma_{\mathsf{timestamp}}(\llbracket t_2 \rrbracket^{\mathbb{W}}_{\mathcal{I}})) \text{ by IH}
\end{aligned}$$

By definition, we have that $\llbracket t_1 \preceq t_2 \rrbracket^{\mathbb{W}}_{\mathcal{I}} = \top$ if, and only if, $0 < \sigma_{\mathsf{timestamp}}(\llbracket t_1 \rrbracket^{\mathbb{W}}_{\mathcal{I}}) \leq \sigma_{\mathsf{timestamp}}(\llbracket t_2 \rrbracket^{\mathbb{W}}_{\mathcal{I}}) \leq \llbracket \mathsf{max_t} \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}$, and this allows us to conclude.

Case $\phi = (t_1 =_s t_2)$ with $s \neq \mathsf{timestamp}$. We have that:

$$\begin{aligned}
&\llbracket \mathsf{O_f}(t_1 =_s t_2)) \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}} \\
&= \llbracket =_s \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}(\llbracket \mathsf{O_t}(t_1) \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}, \llbracket \mathsf{O_t}(t_2) \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}) \\
&= \llbracket =_s \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}(\sigma_s(\llbracket t_1 \rrbracket^{\mathbb{W}}_{\mathcal{I}}), \sigma_s(\llbracket t_2 \rrbracket^{\mathbb{W}}_{\mathcal{I}})) \text{ by IH} \\
&= \llbracket =_s \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}(\llbracket t_1 \rrbracket^{\mathbb{W}}_{\mathcal{I}}, \llbracket t_2 \rrbracket^{\mathbb{W}}_{\mathcal{I}}) \text{ as } s \neq \mathsf{timestamp} \\
&= \llbracket =_s \rrbracket^{\mathbb{W}}_{\mathcal{I}}(\llbracket t_1 \rrbracket^{\mathbb{W}}_{\mathcal{I}}, \llbracket t_2 \rrbracket^{\mathbb{W}}_{\mathcal{I}}) = \llbracket t_1 =_s t_2 \rrbracket^{\mathbb{W}}_{\mathcal{I}}
\end{aligned}$$

Case $\phi = p(t_1, \ldots, t_n)$ with $p \in \Sigma_P$ but not $\preceq$, $\mathsf{happens}$, $=_s$. Assuming that the arity of p is $p(s_1, \ldots, s_n)$, we have that:

$$\begin{aligned}
&\llbracket \mathsf{O_f}(p(t_1, \ldots, t_n)) \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}} \\
&= \llbracket p \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}(\llbracket \mathsf{O_t}(t_1) \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}, \ldots, \llbracket \mathsf{O_t}(t_n) \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}) \\
&= \llbracket p \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}(\sigma_{s_1}(\llbracket t_1 \rrbracket^{\mathbb{W}}_{\mathcal{I}}), \ldots, \sigma_{s_n}(\llbracket t_n \rrbracket^{\mathbb{W}}_{\mathcal{I}})) \text{ by IH} \\
&= \llbracket p \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}(\overline{\sigma}_{s_1}(\sigma_{s_1}(\llbracket t_1 \rrbracket^{\mathbb{W}}_{\mathcal{I}})), \ldots, \overline{\sigma}_{s_n}(\sigma_{s_n}(\llbracket t_n \rrbracket^{\mathbb{W}}_{\mathcal{I}}))) \\
&= \llbracket p \rrbracket^{\mathbb{W}}_{\overline{\mathcal{I}}}(\llbracket t_1 \rrbracket^{\mathbb{W}}_{\mathcal{I}}, \ldots, \llbracket t_n \rrbracket^{\mathbb{W}}_{\mathcal{I}}) \text{ thanks to Lemma 2} \\
&= \llbracket p(t_1, \ldots, t_n) \rrbracket^{\mathbb{W}}_{\mathcal{I}}
\end{aligned}$$

Case $\phi = \mathsf{if}\ \phi_0\ \mathsf{then}\ \phi_1\ \mathsf{else}\ \phi_2$. We have that:

$$\begin{aligned}
&[\![\mathsf{O_f}(\mathsf{if}\ \phi_0\ \mathsf{then}\ \phi_1\ \mathsf{else}\ \phi_2)]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}} \\
&\quad = [\![\mathsf{if}\ \mathsf{O_f}(\phi_0)\ \mathsf{then}\ \mathsf{O_f}(\phi_1)\ \mathsf{else}\ \mathsf{O_f}(\phi_2)]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}} \\
&\quad = \begin{cases} [\![\mathsf{O_f}(\phi_1)]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}} \text{ when } [\![\mathsf{O_f}(\phi_0)]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}} = \top \\ [\![\mathsf{O_f}(\phi_2)]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}} \text{ otherwise} \end{cases} \\
&\quad = \begin{cases} [\![\phi_1]\!]^{\mathsf{W}}_{\mathcal{I}} \text{ when } [\![\phi_0]\!]^{\mathsf{W}}_{\mathcal{I}} = \top \\ [\![\phi_2]\!]^{\mathsf{W}}_{\mathcal{I}} \text{ otherwise} \end{cases} \quad \text{by IH} \\
&\quad = [\![\mathsf{if}\ \phi_0\ \mathsf{then}\ \phi_1\ \mathsf{else}\ \phi_2]\!]^{\mathsf{W}}_{\mathcal{I}} = [\![\phi]\!]^{\mathsf{W}}_{\mathcal{I}}
\end{aligned}$$

Case $\phi = \forall y{:}\mathsf{timestamp}.\phi_0$ *(resp* $\exists y{:}\mathsf{timestamp}.\phi_0$*).* Let $\overline{\mathcal{I}}^+ = \overline{\mathcal{I}}[y \mapsto n]$, and $\mathcal{I}^+ = \mathcal{I}[y \mapsto \overline{\sigma}_{\mathsf{timestamp}}(n)]$. We have that:

$$[\![\mathsf{O_f}(\forall y{:}\mathsf{timestamp}.\phi_0)]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}} = [\![\forall y{:}\mathsf{int}.\mathsf{O_f}(\phi_0)]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}}$$

$$= \begin{cases} \top \text{ when } [\![\mathsf{O_f}(\phi_0)]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}^+} = \top \quad \forall\, n \in \mathbb{Z} \\ \bot \text{ otherwise} \end{cases}$$

$$= \begin{cases} \top \text{ when } [\![\mathsf{O_f}(\phi_0)]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}^+} = \top \quad \forall\, n \in \{0, \ldots, [\![\mathsf{max_t}]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}}\} \\ \bot \text{ otherwise} \end{cases}$$

thanks to Lemma 3

$$= \begin{cases} \top \text{ when } [\![\phi_0]\!]^{\mathsf{W}}_{\mathcal{I}^+} = \top \quad \forall\, n \in \{0, \ldots, [\![\mathsf{max_t}]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}}\} \\ \bot \text{ otherwise} \end{cases}$$

relying on our IH, and on the fact that $\overline{\mathcal{I}}^+ = \overline{\mathcal{I}^+}$ when $n \in \{0, \ldots, [\![\mathsf{max_t}]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}}\}$.

Actually, we have that:

$$[\![\mathsf{timestamp}]\!]^{\mathsf{W}}_{\mathcal{I}} = \{\overline{\sigma}_{\mathsf{timestamp}}(n) \mid n \in \{0, \ldots, [\![\mathsf{max_t}]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}}\}.$$

Hence, we have that:

$$[\![\mathsf{O_f}(\forall y{:}\mathsf{timestamp}.\phi_0)]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}} = [\![\forall y{:}\mathsf{timestamp}.\phi_0]\!]^{\mathsf{W}}_{\mathcal{I}}.$$

Case $\phi = \forall y{:}s.\phi_0$ *(resp* $\exists y{:}s.\phi_0$*) with* $s \neq \mathsf{timestamp}$. Let $\overline{\mathcal{I}}^+ = \overline{\mathcal{I}}[y \mapsto a]$, and $\mathcal{I}^+ = \mathcal{I}[y \mapsto a]$. We have that:

$$[\![\mathsf{O_f}(\forall y{:}s.\phi_0)]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}} = [\![\forall y{:}s.\mathsf{O_f}(\phi_0)]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}}$$

$$= \begin{cases} \top \text{ when } [\![\mathsf{O_f}(\phi_0)]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}^+} = \top \text{ for any } a \in [\![s]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}} \\ \bot \text{ otherwise} \end{cases}$$

$$= \begin{cases} \top \text{ when } [\![\phi_0]\!]^{\mathsf{W}}_{\mathcal{I}^+} = \top \text{ for any } a \in [\![s]\!]^{\mathsf{W}}_{\mathcal{I}} \\ \bot \text{ otherwise} \end{cases}$$

relying on our IH, on the fact that $\overline{\mathcal{I}}^+ = \overline{\mathcal{I}^+}$, and $[\![s]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}} = [\![s]\!]^{\mathsf{W}}_{\mathcal{I}}$

$$= [\![\forall y{:}s.\phi_0]\!]^{\mathsf{W}}_{\mathcal{I}}$$

We now consider the different cases for building a term.

Case t *is a variable of sort* s*, say* x. Applying the definitions, we have that:

$$[\![\mathsf{O_t}(x)]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}} = \sigma_s([\![x]\!]^{\mathsf{W}}_{\mathcal{I}}).$$

Case $t = \mathsf{pred}(t_0)$. We have that:

$$\begin{aligned}
[\![\mathsf{O_t}(\mathsf{pred}(t_0))]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}} &= [\![\mathsf{pred}]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}}([\![\mathsf{O_t}(t_0)]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}}) \\
&= [\![\mathsf{pred}]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}}(\sigma_{\mathsf{timestamp}}([\![t_0]\!]^{\mathsf{W}}_{\mathcal{I}})) \text{ by IH}
\end{aligned}$$

If $\sigma_{\mathsf{timestamp}}([\![t_0]\!]^{\mathsf{W}}_{\mathcal{I}}) = 0$, then we have that $[\![t_0]\!]^{\mathsf{W}}_{\mathcal{I}} = u$, and we have that $[\![\mathsf{O_t}(\mathsf{pred}(t_0))]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}} = \sigma_{\mathsf{timestamp}}([\![\mathsf{pred}(t_0)]\!]^{\mathsf{W}}_{\mathcal{I}}) = 0$ relying on our definitions.

Otherwise, $\sigma_{\mathsf{timestamp}}([\![t_0]\!]^{\mathsf{W}}_{\mathcal{I}}) \in \{1, \ldots, [\![\mathsf{max_t}]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}}\}$, and in that case, we have that

$$[\![\mathsf{O_t}(\mathsf{pred}(t_0))]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}} = \sigma_{\mathsf{timestamp}}([\![t_0]\!]^{\mathsf{W}}_{\mathcal{I}}) - 1.$$

Actually, for any $t \in [\![\mathsf{timestamp}]\!]^{\mathsf{W}}_{\mathcal{I}}$ such that $[\![\mathsf{happens}(t)]\!]^{\mathsf{W}}_{\mathcal{I}} = \top$, we have that:

$$\begin{aligned}
&\#\{t' \in [\![\mathsf{timestamp}]\!]^{\mathsf{W}} \mid [\![\leq]\!]^{\mathsf{W}}_{\mathcal{I}}(t', t)\} - 1 \\
= \ &\#\{t' \in [\![\mathsf{timestamp}]\!]^{\mathsf{W}} \mid [\![\leq]\!]^{\mathsf{W}}_{\mathcal{I}}(t', \mathsf{pred}(t))\}.
\end{aligned}$$

We know that $[\![\mathsf{happens}(t_0)]\!]^{\mathsf{W}}_{\mathcal{I}} = \top$, and thus we have that:

$$[\![\mathsf{O_t}(\mathsf{pred}(t_0))]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}} = \sigma_{\mathsf{timestamp}}([\![\mathsf{pred}(t_0)]\!]^{\mathsf{W}}_{\mathcal{I}}).$$

Case $t = f(t_1, \ldots, t_n)$ *with* $f(s_1, \ldots, s_n) : s \in \Sigma_F \setminus \{\mathsf{pred}\}$. Relying on our induction hypothesis, and Lemma 2, we have that:

$$\begin{aligned}
&[\![\mathsf{O_t}(f(t_1, \ldots, t_n))]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}} \\
&= \sigma_s([\![f]\!]^{\mathsf{W}}_{\mathcal{I}}(\overline{\sigma}_{s_1}([\![\mathsf{O_t}(t_1)]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}}), \ldots, \overline{\sigma}_{s_n}([\![\mathsf{O_t}(t_n)]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}}))) \\
&= \sigma_s([\![f]\!]^{\mathsf{W}}_{\mathcal{I}}(\overline{\sigma}_{s_1}(\sigma_{s_1}([\![t_1]\!]^{\mathsf{W}}_{\mathcal{I}})), \ldots, \overline{\sigma}_{s_n}(\sigma_{s_n}([\![t_n]\!]^{\mathsf{W}}_{\mathcal{I}})))) \\
&= \sigma_s([\![f]\!]^{\mathsf{W}}_{\mathcal{I}}([\![t_1]\!]^{\mathsf{W}}_{\mathcal{I}}, \ldots, [\![t_n]\!]^{\mathsf{W}}_{\mathcal{I}})) \\
&= \sigma_s([\![f(t_1, \ldots, t_n)]\!]^{\mathsf{W}}_{\mathcal{I}})
\end{aligned}$$

Case $t = \mathsf{if}\ \phi\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2$ *with* t_1 *and* t_2 *of sort* s. We have that:

$$\begin{aligned}
&[\![\mathsf{O_t}(\mathsf{if}\ \phi\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2)]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}} \\
&\quad = [\![\mathsf{if}\ \mathsf{O_f}(\phi)\ \mathsf{then}\ \mathsf{O_t}(t_1)\ \mathsf{else}\ \mathsf{O_t}(t_2)]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}} \\
&\quad = \begin{cases} [\![\mathsf{O_t}(t_1)]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}} \text{ when } [\![\mathsf{O_f}(\phi)]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}} = \top \\ [\![\mathsf{O_t}(t_2)]\!]^{\mathsf{W}}_{\overline{\mathcal{I}}} \text{ otherwise} \end{cases} \\
&\quad = \begin{cases} \sigma_s([\![t_1]\!]^{\mathsf{W}}_{\mathcal{I}}) \text{ when } [\![\phi]\!]^{\mathsf{W}}_{\mathcal{I}} = \top \\ \sigma_s([\![t_2]\!]^{\mathsf{W}}_{\mathcal{I}}) \text{ otherwise} \end{cases} \\
&\qquad \text{applying the induction hypothesis} \\
&\quad = \sigma_s([\![\mathsf{if}\ \phi\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2]\!]^{\mathsf{W}}_{\mathcal{I}}) = \sigma_s([\![t]\!]^{\mathsf{W}}_{\mathcal{I}})
\end{aligned}$$

This concludes the proof. □