Canonical Narrowing with Irreducibility and SMT Constraints as a Generic Symbolic Protocol Analysis Method

Raúl López-Rueda(✉) and Santiago Escobar

VRAIN, Universitat Politècnica de València, Valencia, Spain
{rloprue,sescobar}@upv.es

Abstract. Nowadays, formal cryptographic protocol analysis relies on symbolic techniques such as narrowing and equational unification, e.g. Maude-NPA, Tamarin or AKISS crypto tools. In previous works, we developed a new narrowing strategy, called canonical narrowing, which manages to reduce the state explosion problem by introducing irreducibility constraints. In this paper, we extend canonical narrowing to handle conditional rules with SMT constraints. We demonstrate the viability of this method with the Brands and Chaum protocol using time and location information described as SMT constraints on the real numbers.

Keywords: Canonical narrowing · SMT solver · Maude · Security protocols · Brands and Chaum

1 Introduction

Formal protocol analysis allows to determine whether an attacker can cause a protocol to fail any of its security objectives. One of the ways to perform this type of analysis is through the use of symbolic techniques, such as narrowing. There are tools for protocol analysis, like Maude-NPA [8], that use narrowing together with equational unification as a basis. These techniques are efficiently supported by the Maude language, and are also used in other protocol analysis tools such as Tamarin [15] or AKISS [4]. In our works [10,14], we already developed a new narrowing algorithm, called canonical narrowing, which manages to reduce the state explosion problem by introducing irreducibility constraints.

In a large number of protocols, the use of laws of physics that use real numbers to represent distances, time, or coordinates is essential. The formal analysis of this type of protocols can be done using either an explicit model with physical information, or by using an abstract model without physical information, e.g.,

This work has been partially supported by the EC H2020-EU grant agreement No. 952215 (TAILOR), by the grant RTI2018-094403-B-C32 funded by MCIN/AEI/10.13039/501100011033 and ERDF "A way of making Europe", by the grant PROMETEO/2019/098 funded by Generalitat Valenciana, and by the grant PCI2020-120708-2 funded by MICIN/AEI/10.13039/501100011033 and by the European Union NextGenerationEU/PRTR.

K. Bae (Ed.): WRLA 2022, LNCS 13252, pp. 45–64, 2022.
https://doi.org/10.1007/978-3-031-12441-9_3

untimed, and showing it is sound and complete with respect to a model with physical information. The former is more intuitive for the user, but the latter is often chosen because not all cryptographic protocol analysis tools support reasoning about, e.g., time or space. SMT solvers allow precisely the use of explicit models with physical information, translating the physical laws into SMT constraints. In order to analyze these models using narrowing algorithms, there is a need to extend them so that they are capable of handling these restrictions. One way to do it is by having narrowing to handle conditional rules, as in [19], in which each of the constraints will be collected at runtime. In the following example, we show one of the protocols that use laws of physics. This protocol really goes beyond existing narrowing approaches such as [10,14,19], since two cryptographic primitives are combined, exclusive-or over a set of nonces and a commitment scheme, apart of time and location represented as real numbers, requiring both irreducibility and SMT constraints.

Example 1. The Brands-Chaum protocol [3] specifies communication between a verifier V and a prover P. P needs to authenticate itself to V, and also needs to prove that it is within a distance "d" of it. A typical interaction between the prover and the verifier is as follows, where N_A denotes a nonce generated by A, S_A denotes a secret generated by A, $X; Y$ denotes concatenation of two messages X and Y, $commit(N, S)$ denotes commitment of secret S with a nonce N, $open(N, S, C)$ denotes opening a commitment C using the nonce N and checking whether it carries the secret S, $\oplus$ is the exclusive-or operator, and $sign(A, M)$ denotes A signing message M.

$P \rightarrow V : commit(N_P, S_P)$
//The prover sends his name and a commitment
$V \rightarrow P : N_V$
//The verifier sends a nonce and records the time when this message was sent
$P \rightarrow V : N_P \oplus N_V$
//The verifier checks the answer message arrives within two times a fixed distance
$P \rightarrow V : S_P$
//The prover sends the committed secret and the verifier opens the commitment
$P \rightarrow V : sign_P(N_V; N_P \oplus N_V)$
//The prover signs the two rapid exchange messages

We assume the participants are located at an arbitrary given topology (participants do not move from their assigned locations) with distance constraints, where travelled time and coordinates are represented by a real number. We assumed coordinates P_x, P_y, P_z for each participant P.

The previous informal Alice&Bob notation was naturally extended to include time in [1] and to include both time and location in [2]. First, we add the time when a message was sent or received as a subindex $P_{t_1} \rightarrow V_{t_2}$. Second, the sending and receiving times of a message differ by the distance between them just by adding some location constraints

$$\lfloor d(A,B) \rfloor := (d(A,B) \geq 0 \wedge d(A,B)^2 = (A_x - B_x)^2 + (A_y - B_y)^2 + (A_z - B_z)^2)$$

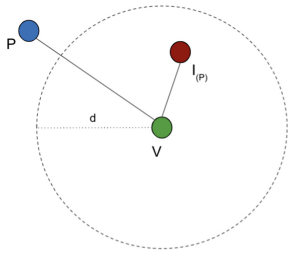

Fig. 1. Mafia Attack

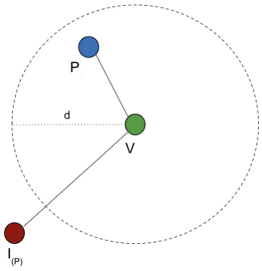

Fig. 2. Hijacking Attack

Third, the distance bounding constraint of the verifier is represented as an arbitrary distance d. Time and space constraints are written using quantifier-free formulas in real arithmetic. For convenience, we allow both $2 * x = x + x$ and the monus function $x \dot{-} y = \mathit{if}\ y < x\ \mathit{then}\ x - y\ \mathit{else}\ 0$ as definitional extensions.

Example 2 (Cont'd Example 1). In the following time and space sequence of actions, a vertical bar differentiates between the process and corresponding constraints associated to the metric space. The following action sequence differs from [1] only on the terms $\lfloor d(P,V) \rfloor$.

$$
\begin{array}{rll}
P_{t_1} \to V_{t'_1}: & \mathit{commit}(N_P, S_P) & \mid t'_1 = t_1 + d(P,V) \wedge \lfloor d(P,V) \rfloor \\
V_{t_2} \to P_{t'_2}: & N_V & \mid t'_2 = t_2 + d(P,V) \wedge t_2 \geq t'_1 \wedge \lfloor d(P,V) \rfloor \\
P_{t_3} \to V_{t'_3}: & N_P \oplus N_V & \mid t'_3 = t_3 + d(P,V) \wedge t_3 \geq t'_2 \wedge \lfloor d(P,V) \rfloor \\
V: & t'_3 \dot{-} t_2 \leq 2 * d & \\
P_{t_4} \to V_{t'_4}: & S_P & \mid t'_4 = t_4 + d(P,V) \wedge t_4 \geq t_3 \wedge \lfloor d(P,V) \rfloor \\
V: & \mathit{open}(N_P, S_P, \mathit{commit}(N_P, S_P)) & \\
P_{t_5} \to V_{t'_5}: & \mathit{sign}_P(N_V; N_P \oplus N_V) & \mid t'_5 = t_5 + d(P,V) \wedge t_5 \geq t_4 \wedge \lfloor d(P,V) \rfloor
\end{array}
$$

The Brands-Chaum protocol is designed to defend against mafia frauds, where an honest prover is outside the neighborhood of the verifier (i.e., $d(P,V) > d$) but an intruder is inside (i.e., $d(I,V) \leq d$), pretending to be the honest prover as depicted in Fig. 1. The following is an example of an *attempted* mafia fraud, in which the intruder simply forwards messages back and forth between the prover and the verifier. We write $I(P)$ to denote an intruder pretending to be an honest prover P.

$$
\begin{array}{rll}
P_{t_1} \to I_{t_2} & : \mathit{commit}(N_P, S_P) & \mid t_2 = t_1 + d(P,I) \wedge \lfloor d(P,I) \rfloor \\
I(P)_{t_2} \to V_{t_3} & : \mathit{commit}(N_P, S_P) & \mid t_3 = t_2 + d(V,I) \wedge \lfloor d(V,I) \rfloor \\
V_{t_3} \to I(P)_{t_4} & : N_V & \mid t_4 = t_3 + d(V,I) \wedge \lfloor d(V,I) \rfloor \\
I_{t_4} \to P_{t_5} & : N_V & \mid t_5 = t_4 + d(P,I) \wedge \lfloor d(P,I) \rfloor \\
P_{t_5} \to I_{t_6} & : N_P \oplus N_V & \mid t_6 = t_5 + d(P,I) \wedge \lfloor d(P,I) \rfloor \\
I(P)_{t_6} \to V_{t_7} & : N_P \oplus N_V & \mid t_7 = t_6 + d(V,I) \wedge \lfloor d(V,I) \rfloor \\
V & : t_7 \dot{-} t_3 \leq 2 * d & \\
P_{t_8} \to I_{t_9} & : S_P & \mid t_9 = t_8 + d(P,I) \wedge t_8 \geq t_5 \wedge \lfloor d(P,I) \rfloor \\
I(P)_{t_{10}} \to V_{t_{11}} & : S_P & \mid t_{11} = t_{10} + d(V,I) \wedge t_{11} \geq t_7 \wedge \lfloor d(V,I) \rfloor \\
I(P)_{t_{12}} \to V_{t_{13}} & : \mathit{sign}_P(N_V; N_P \oplus N_V) & \mid t_{13} = t_{12} + d(V,I) \wedge t_{13} \geq t_{11} \wedge \lfloor d(V,I) \rfloor
\end{array}
$$

This attack is physically unfeasible, since it would require that $2 * d(V, I) + 2 * d(P, I) \leq 2 * d$, which is unsatisfiable by $d(V, P) > d > 0$ and the triangular inequality $d(V, P) \leq d(V, I) + d(P, I)$, satisfied in three-dimensional space. This attack was already unfeasible in [1] using only the metric space assumptions and in [2] using a Euclidean space.

However, a distance hijacking attack is possible (i.e., the time and distance constraints are satisfiable), as depicted in Fig. 2, where an intruder located outside the neighborhood of the verifier (i.e., $d(V, I) > d$) succeeds in convincing the verifier that he is inside the neighborhood by exploiting the presence of an honest prover in the neighborhood (i.e., $d(V, P) \leq d$) to achieve his goal. The following is an example of a *successful* distance hijacking, in which the intruder listens to the exchanged messages between the prover and the verifier but builds the last message.

$$
\begin{array}{rll}
P_{t_1} \rightarrow V_{t_2} & : \mathit{commit}(N_P, S_P) & \mid t_2 = t_1 + d(P,V) \wedge \lfloor d(P,V) \rfloor \\
V_{t_2} \rightarrow P_{t_3}, I_{t'_3} & : N_V & \mid t_3 = t_2 + d(P,V) \wedge \lfloor d(P,V) \rfloor \\
 & & \mid t'_3 = t_2 + d(I,V) \wedge \lfloor d(V,I) \rfloor \\
P_{t_3} \rightarrow V_{t_4}, I_{t'_4} & : N_P \oplus N_V & \mid t_4 = t_3 + d(P,V) \wedge \lfloor d(P,V) \rfloor \\
 & & \mid t'_4 = t_3 + d(I,P) \wedge \lfloor d(I,P) \rfloor \\
V & : t_4 \dot{-} t_2 \leq 2 * d & \\
P_{t_5} \rightarrow V_{t_6} & : S_P & \mid t_6 = t_5 + d(P,V) \wedge \lfloor d(P,V) \rfloor \\
 & & \mid t_5 \geq t_3 \wedge t_6 \geq t_4 \\
I(P)_{t_7} \rightarrow V_{t_8} & : \mathit{sign}_I(N_V; N_P \oplus N_V) & \mid t_8 = t_7 + d(I,V) \wedge \lfloor d(I,V) \rfloor \\
 & & \mid t_7 \geq t'_4 \wedge t_8 \geq t_6
\end{array}
$$

This attack was feasible in [1] using the metric space assumptions, and it was also possible in three-dimensional space in [2].

In Sect. 2, we provide some preliminaries. In Sect. 3, we introduce our new canonical narrowing with irreducibility and SMT constraints. In Sect. 4, we describe our implementation. In Sect. 5, we present some experiments using the Brands and Chaum protocol that prove its viability. In Sect. 6, we conclude and give some future work.

2 Preliminaries

We follow the classical notation and terminology from [21] for term rewriting, and from [16,19] for rewriting logic and order-sorted notions.

We assume an order-sorted signature Σ with a poset of sorts $(S, \leq)$. The poset $(\mathsf{S}, \leq)$ of sorts for Σ is partitioned into equivalence classes, called *connected components*, by the equivalence relation $(\leq \cup \geq)^+$. We assume that each connected component $[\mathsf{s}]$ has a *top element* under $\leq$, denoted $\top_{[\mathsf{s}]}$ and called the *top sort* of $[\mathsf{s}]$. This involves no real loss of generality, since if $[\mathsf{s}]$ lacks a top sort, it can be easily added.

We assume an S-sorted family $\mathcal{X} = \{\mathcal{X}_\mathsf{s}\}_{\mathsf{s} \in \mathsf{S}}$ of disjoint variable sets with each $\mathcal{X}_\mathsf{s}$ countably infinite. $\mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ is the set of terms of sort s, and $\mathcal{T}_{\Sigma,\mathsf{s}}$ is the

set of ground terms of sort s. We write $\mathcal{T}_\Sigma(\mathcal{X})$ and $\mathcal{T}_\Sigma$ for the corresponding order-sorted term algebras. Given a term t, $Var(t)$ denotes the set of variables in t.

A *substitution* $\sigma \in \mathcal{S}ubst(\Sigma, \mathcal{X})$ is a sorted mapping from a finite subset of $\mathcal{X}$ to $\mathcal{T}_\Sigma(\mathcal{X})$. Substitutions are written as $\sigma = \{X_1 \mapsto t_1, \ldots, X_n \mapsto t_n\}$ where the domain of σ is $Dom(\sigma) = \{X_1, \ldots, X_n\}$ and the set of variables introduced by terms $t_1, \ldots, t_n$ is written $Ran(\sigma)$. The identity substitution is *id*. Substitutions are homomorphically extended to $\mathcal{T}_\Sigma(\mathcal{X})$. The application of substitution σ to a term t is denoted by $t\sigma$ or $\sigma(t)$.

A Σ-*equation* is an unoriented pair $t = t'$, where $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ for some sort $\mathsf{s} \in \mathsf{S}$. Given Σ and a set E of Σ-equations, order-sorted equational logic induces a congruence relation $=_E$ on terms $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})$ (see [17]). Throughout this paper we assume that $\mathcal{T}_{\Sigma,\mathsf{s}} \neq \emptyset$ for every sort s, because this affords a simpler deduction system. We write $\mathcal{T}_{\Sigma/E}(\mathcal{X})$ and $\mathcal{T}_{\Sigma/E}$ for the corresponding order-sorted term algebras modulo the congruence closure $=_E$, denoting the equivalence class of a term $t \in \mathcal{T}_\Sigma(\mathcal{X})$ as $[t]_E \in \mathcal{T}_{\Sigma/E}(\mathcal{X})$.

The first-order language of equational Σ-formulas is defined as: Σ-equations $t = t'$ as basic atoms, conjunction $\wedge$ of formulas, disjunction $\vee$ of formulas, negation $\neg$ of a formula, universal quantification $\forall$ of a variable $x{:}\mathsf{s}$ in a formula, and existential quantification $\exists$ of a variable $x{:}\mathsf{s}$ in a formula. A formula is quantifier-free (QF) if it does not contain any quantifier. Given a Σ-algebra A, a formula φ, and an assignment $\alpha \in X \mapsto A$ for the free variables X in φ, $A, \alpha \models \varphi$ denotes that φ is satisfied and $A \models \varphi$ holds if $\forall \alpha : A, \alpha \models \varphi$.

An *equational theory* (Σ, E) is a pair with Σ an order-sorted signature and E a set of Σ-equations. An equational theory (Σ, E) is *regular* if for each $t = t'$ in E, we have $Var(t) = Var(t')$. An equational theory (Σ, E) is *linear* if for each $t = t'$ in E, each variable occurs only once in t and in t'. An equational theory (Σ, E) is *sort-preserving* if for each $t = t'$ in E, each sort s, and each substitution σ, we have $t\sigma \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ iff $t'\sigma \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$. An equational theory (Σ, E) is *defined using top sorts* if for each equation $t = t'$ in E, all variables in $Var(t)$ and $Var(t')$ have a top sort. Given two equational theories $G = (\Sigma, E)$ and $T = (\Delta, \Gamma)$, we say T is the background theory of E iff $\Sigma \subseteq \Delta$ and for each ground Σ-formula φ, $\mathcal{T}_{\Sigma/E} \models \varphi \iff T \models \varphi$.

An E-*unifier* for a Σ-equation $t = t'$ is a substitution σ such that $t\sigma =_E t'\sigma$. For $Var(t) \cup Var(t') \subseteq W$, a set of substitutions $CSU_E^W(t = t')$ is said to be a *complete* set of unifiers for the equality $t = t'$ modulo E away from W iff: (i) each $\sigma \in CSU_E^W(t = t')$ is an E-unifier of $t = t'$; (ii) for any E-unifier ρ of $t = t'$ there is a $\sigma \in CSU_E^W(t = t')$ such that $\sigma|_W \sqsupseteq_E \rho|_W$ (i.e., there is a substitution η such that $(\sigma\eta)|_W =_E \rho|_W$); and (iii) for all $\sigma \in CSU_E^W(t = t')$, $Dom(\sigma) \subseteq (Var(t) \cup Var(t'))$ and $Ran(\sigma) \cap W = \emptyset$.

A *conditional rewrite rule* is an oriented pair $l \to r$ if φ, where $l \notin \mathcal{X}$, φ is a QF Σ-formula, and $l, r \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ for some sort $\mathsf{s} \in \mathsf{S}$. An unconditional rewrite rule is written $l \to r$. A *conditional order-sorted rewrite theory* is a triple (Σ, E, R, T) with Σ an order-sorted signature, E a set of Σ-equations, T is a background theory, and R a set of conditional rewrite rules. The set R of rules

is *sort-decreasing* if for each $t \to t'$ (or $t \to t'$ if φ) in R, each $\mathsf{s} \in \mathsf{S}$, and each substitution σ, $t'\sigma \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ implies $t\sigma \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$.

The rewriting relation on $\mathcal{T}_\Sigma(\mathcal{X})$, written $t \to_R t'$ or $t \to_{p,R} t'$ holds between t and t' iff there exist a $p \in Pos_\Sigma(t)$, $l \to r$ if $\varphi \in R$ and a substitution σ, such that $T \models \varphi\sigma$, $t|_p = l\sigma$, and $t' = t[r\sigma]_p$. The relation $\to_{R/E}$ on $\mathcal{T}_\Sigma(\mathcal{X})$ is $=_E;\to_R;=_E$. The transitive (resp. transitive and reflexive) closure of $\to_{R/E}$ is denoted $\to^+_{R/E}$ (resp. $\to^*_{R/E}$). A term t is called $\to_{R/E}$-irreducible (or just R/E-irreducible) if there is no term t' such that $t \to_{R/E} t'$. For $\to_{R/E}$ confluent and terminating, the irreducible version of a term t is denoted by $t{\downarrow_{R/E}}$.

A relation $\to_{R,E}$ on $\mathcal{T}_\Sigma(\mathcal{X})$ is defined as: $t \to_{p,R,E} t'$ (or just $t \to_{R,E} t'$) iff there are a position $p \in Pos_\Sigma(t)$, a rule $l \to r$ if φ in R, and a substitution σ such that $T \models \varphi\sigma$, $t|_p =_E l\sigma$ and $t' = t[r\sigma]_p$. Reducibility of $\to_{R/E}$ is undecidable in general since E-congruence classes can be arbitrarily large. Therefore, R/E-rewriting is usually implemented [13] by R,E-rewriting under some conditions on R and E such as confluence, termination, and coherence.

We call (Σ, B, E) a *decomposition* of an order-sorted equational theory $(\Sigma, E \cup B)$ if B is regular, linear, sort-preserving, defined using top sorts, and has a finitary and complete unification algorithm, which implies that B-matching is decidable, and the equations E oriented into rewrite rules $\overrightarrow{E}$ are *convergent*, i.e., confluent, terminating, and strictly coherent [18] modulo B, and sort-decreasing.

Given a decomposition (Σ, B, E) of an equational theory, (t', θ) is an E,B-*variant* [6,11] (or just a variant) of term t if $t\theta{\downarrow_{E,B}} =_E t'$ and $\theta{\downarrow_{E,B}} =_E \theta$. A *complete set of* E,B*-variants* [11] (up to renaming) of a term t is a subset, denoted by $[\![t]\!]_{E,B}$, of the set of all E,B-variants of t such that, for each E,B-variant (t', σ) of t, there is an E,B-variant $(t'', \theta) \in [\![t]\!]_{E,B}$ such that $(t'', \theta) \sqsupseteq_{E,B} (t', \sigma)$, i.e., there is a substitution ρ such that $t' =_B t''\rho$ and $\sigma|_{Var(t)} =_B (\theta\rho)|_{Var(t)}$. A decomposition (Σ, B, E) has the *finite variant property* (FVP) [11] (also called a *finite variant decomposition*) iff for each Σ-term t, a complete set $[\![t]\!]_{E,B}$ of its most general variants is finite.

In what follows, the set G of equations will in practice be $G = E \uplus B$ and will have a decomposition (Σ, B, E).

Definition 1 (Reachability goal). *Given an order-sorted rewrite theory* (Σ, G, R, T)*, a* reachability goal *is defined as a pair* $t \stackrel{?}{\to}^*_{R/G} t'$*, where* $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$*. It is abbreviated as* $t \stackrel{?}{\to}^* t'$ *when the theory is clear from the context;* t *is the* source *of the goal and* t' *is the* target*. A substitution* σ *is a* R/G-solution *of the reachability goal (or just a solution for short) iff there is a sequence* $\sigma(t) \to_{R/G} \sigma(u_1) \to_{R/G} \cdots \to_{R/G} \sigma(u_{k-1}) \to_{R/G} \sigma(t')$*.*

A set Γ *of substitutions is said to be a* complete set of solutions *of* $t \stackrel{?}{\to}^*_{R/G} t'$ *iff (i) every substitution* $\sigma \in \Gamma$ *is a solution of* $t \stackrel{?}{\to}^*_{R/G} t'$*, and (ii) for any solution* ρ *of* $t \stackrel{?}{\to}^*_{R/G} t'$*, there is a substitution* $\sigma \in \Gamma$ *more general than* ρ *modulo* G*, i.e.,* $\sigma|_{Var(t) \cup Var(t')} \sqsupseteq_G \rho|_{Var(t) \cup Var(t')}$*.*

This provides a tool-independent semantic framework for symbolic reachability analysis of protocols under algebraic properties. Note that we have removed the condition $Var(\varphi) \cup Var(r) \subseteq Var(l)$ for rewrite rules $l \to r$ if $\varphi \in R$ and thus a solution of a reachability goal must be applied to all terms in the rewrite sequence. If the terms t and t' in a goal $t \stackrel{?}{\to}^*_{T/G} t'$ are ground and rules have no extra variables in their right-hand sides, then goal solving becomes a standard rewriting reachability problem. However, since we allow terms t, t' with variables, we need a mechanism more general than standard rewriting to find solutions of reachability goals. *Narrowing* with R modulo G generalizes rewriting by performing *unification* at non-variable positions instead of the usual matching modulo G. Soundness and completeness of narrowing for solving reachability goals are proved in [13,20] for unconditional rules R modulo an equational theory G and in [19] for conditional rules R modulo an equational theory G, both with the restriction of considering only order-sorted *topmost* rewrite theories, i.e., rewrite theories were all the rewrite steps happen at the top of the term.

3 Canonical Narrowing with Irreducibility and SMT Constraints

This section extends the canonical narrowing strategy of [10] with SMT constraints.

When $(\Sigma, E \cup B)$ has a decomposition as (Σ, B, E), then the initial algebra $\mathcal{T}_{\Sigma/E\cup B}$ is isomorphic to the canonical term algebra $\mathcal{C}_{\Sigma/E\cup B} = (C_{\Sigma/E\cup B}, \to_{R/E\cup B})$, where $C_{\Sigma/E\cup B} = \{C_{\Sigma/E\cup B,\mathsf{s}}\}_{\mathsf{s}\in\mathsf{S}}$ and $C_{\Sigma/E\cup B,\mathsf{s}} = \{[t{\downarrow_{\overrightarrow{E},B}}]_B \in T_{\Sigma/B} \mid t{\downarrow_{\overrightarrow{E},B}} \in T_{\Sigma,\mathsf{s}}\}$ and where for each $f \in \Sigma$, $f_{\mathcal{C}_{\Sigma/E\cup B}}([t_1]_B, \ldots, [t_n]_B) = [f(t_1, \ldots, t_n){\downarrow_{\overrightarrow{E},B}}]_B$.

We have an isomorphism of initial algebras $\mathcal{T}_{\Sigma/E\cup B} \cong \mathcal{C}_{\Sigma/E\cup B}$. Likewise, we have an isomorphism of free $(\Sigma, E \cup B)$-algebras $\mathcal{T}_{\Sigma/E\cup B}(\mathcal{X}) \cong \mathcal{C}_{\Sigma/E\cup B}(\mathcal{X})$, where $\mathcal{C}_{\Sigma/E\cup B}(\mathcal{X}) = (C_{\Sigma/E\cup B}(\mathcal{X}), \to_{R/E\cup B})$ and

$$C_{\Sigma/E\cup B,\mathsf{s}}(\mathcal{X}) = \{[t{\downarrow_{\overrightarrow{E},B}}]_B \in T_{\Sigma/B}(\mathcal{X}) \mid t{\downarrow_{\overrightarrow{E},B}} \in T_\Sigma(\mathcal{X})_\mathsf{s}\}.$$

The key point of canonical rewriting is that we can simulate rewritings $[t]_{E\cup B} \to_{R/E\cup B} [t']_{E\cup B}$ by corresponding rewritings $[t{\downarrow_{\overrightarrow{E},B}}]_B \to_{R/E\cup B} [t'{\downarrow_{\overrightarrow{E},B}}]_B$ and make rewriting decidable when $(\Sigma, B, \overrightarrow{E})$ is FVP.

Definition 2 (Canonical Rewriting). *Let $\mathcal{R} = (\Sigma, E \cup B, R, T)$ be a topmost order-sorted rewrite theory such that $(\Sigma, E \cup B)$ has an FVP decomposition (Σ, B, E). Let $\mathcal{C}^\circ_{\Sigma/E\cup B}(\mathcal{X})_{\mathsf{State}} = \bigcup \mathcal{C}_{\Sigma/E\cup B}(\mathcal{X})_{\mathsf{State}}$, i.e., $\mathcal{C}^\circ_{\Sigma/E\cup B}(\mathcal{X})_{\mathsf{State}} = \{t{\downarrow_{\overrightarrow{E},B}} \mid t{\downarrow_{\overrightarrow{E},B}} \in T_\Sigma(\mathcal{X})_{\mathsf{State}}\}$, so that $\mathcal{C}^\circ_{\Sigma/E\cup B}(\mathcal{X})_{\mathsf{State}} \subseteq T_\Sigma(\mathcal{X})_{\mathsf{State}}$. We then define the $\to_{R/E,B}$ canonical rewrite relation with rules R modulo $E \cup B$ as the following binary relation $\to_{R/E,B} \subseteq \mathcal{C}^\circ_{\Sigma/E\cup B}(\mathcal{X})_{\mathsf{State}} \times \mathcal{C}^\circ_{\Sigma/E\cup B}(\mathcal{X})_{\mathsf{State}}$, where $t \to_{R/E,B} t'$ iff $\exists l \to r$ if $\varphi \in R$ and $\exists\theta$ with $Dom(\theta) \subseteq Var(l) \cup Var(r) \cup Var(\varphi)$ and $\theta = \theta{\downarrow_{\overrightarrow{E},B}}$ such that: (i) $T \models \varphi\theta$, (ii) $(l\theta){\downarrow_{\overrightarrow{E},B}} =_{E\cup B} t$, and (iii) $t' =_B (r\theta){\downarrow_{\overrightarrow{E},B}}$.*

The claim that $\rightarrow_{R/E,B}$ exactly captures/bisimulates the $\rightarrow_{R/E\cup B}$ rewrite relation is justified by the following result.

Theorem 1. *For each* $t, t' \in T_\Sigma(\mathcal{X})_{\mathsf{State}}$, $t \rightarrow_{R/E\cup B} t'$ *iff* $t{\downarrow_{\overrightarrow{E},B}} \rightarrow_{R/E,B} t'{\downarrow_{\overrightarrow{E},B}}$.

A term $t(x_1{:}s_1, \ldots, x_n{:}s_n)$ can be viewed as a symbolic, effective method to describe a (typically infinite) set of terms, namely the set

$$\lceil t(x_1{:}s_1, \ldots, x_n{:}s_n)\rceil = \{t(u_1, \ldots, u_n) \mid u_i \in \mathcal{T}_\Sigma(\mathcal{X})_{\mathsf{s_i}}\} = \{t\theta \mid \theta \in \mathcal{S}ubst(\Sigma, \mathcal{X})\}.$$

We think as t as a *pattern*, which symbolically describes all its *instances* (including non-ground). However, since (Σ, B, E) is a decomposition of an equational theory $(\Sigma, E \cup B)$, we can consider only normalized instances of t

$$\lceil t\rceil_{\overrightarrow{E},B} = \{(t\theta){\downarrow_{\overrightarrow{E},B}} \mid \theta \in \mathcal{S}ubst(\Sigma, \mathcal{X})\}$$

However, since we are interested in terms that may satisfy some irreducibility and SMT conditions, we can obtain a more expressive symbolic pattern language where patterns are *constrained by both irreducibility and SMT constraints*. That is, we consider constrained patterns of the form $\langle t, \Pi, \varphi\rangle$ where Π is a finite set of normalized terms and φ is a QF Σ-formula. Then we can define:

$$\lceil\langle t, (u_1, \ldots, u_k), \varphi\rangle\rceil_{\overrightarrow{E},B} = \{(t\theta){\downarrow_{\overrightarrow{E},B}} \mid \theta \in \mathcal{S}ubst(\Sigma, \mathcal{X}), T \models \varphi\theta, \\ u_1\theta, \ldots, u_k\theta \text{ are } \overrightarrow{E}, B\text{-normalized}\}.$$

The canonical narrowing relation $\rightsquigarrow_{R/E,B}$ includes irreducibility constraints only for the left-hand sides of the rules and SMT constraints only from the conditional part of the rules.

Definition 3 (Canonical Narrowing). *Given a topmost order-sorted rewrite theory* $(\Sigma, E \cup B, R, T)$ *such that* (Σ, B, E) *is a decomposition of* $(\Sigma, E \cup B)$, *the* canonical narrowing relation with irreducibility constraints *holds between* $\langle t, \Pi, \varphi\rangle$ *and* $\langle t', \Pi', \varphi'\rangle$, *denoted*

$$\langle t, \Pi, \varphi\rangle \rightsquigarrow_{\alpha,R/E,B} \langle t', \Pi', \varphi'\rangle$$

iff there exists $l \rightarrow r$ *if* $\varphi'' \in R$, *which we always assume renamed, so that* $Var(\langle t, \Pi, \varphi\rangle) \cap (Var(r) \cup Var(l) \cup Var(\varphi'')) = \emptyset$, *and a unifier* $\alpha \in CSU^W_{E\cup B}(t = l)$, *where* $W = Var(\langle t, \Pi, \varphi\rangle) \cup Var(r) \cup Var(l) \cup Var(\varphi'')$, *and*

1. $\langle t', \Pi', \varphi'\rangle = \langle r\alpha, \Pi\alpha \cup \{(l\alpha){\downarrow_{\overrightarrow{E},B}}\}, \varphi\alpha \wedge \varphi''\alpha\rangle$,
2. $\Pi\alpha \cup \{(l\alpha){\downarrow_{\overrightarrow{E},B}}\}$ *are* $\overrightarrow{E}, B$*-irreducible, and*
3. φ' *is satisfiable, i.e.,* $\exists\alpha'$ *s.t.* $T \models \varphi'\alpha'$.

Note that we do not require a narrowing step to compute $CSU_{E\cup B}(t=l)$ anymore, we perform regular equational unification but impose an irreducibility constraint on the normal form of the instantiated left-hand side, which can be handled in Maude by using asymmetric unification [7], i.e., equational unification is done with irreducibility constraints.

Irreducibility constraints are computed by using the normalized left-hand side of the rules that are used in the narrowing steps. SMT constraints are simply added to the third component and check for satisfiability. Note that we assume that satisfiability of QF Σ-formulas is decidable, indeed for a subsignature $\Sigma_0 \subseteq \Sigma$ associated to the background theory T. Maude is using the CVC4 SMT solver for satisfiability.

Each trace will carry a different set of irreducibility and SMT constraints, although some of the conditions are shared by having common predecessor nodes. In each new narrowing step, the list of irreducibility constraints computed previously in that sequence must be taken into account, so that if it is necessary to reduce one of the terms appearing in the list to compute a new step, it will be discarded. Similary, the SMT formula carried along the sequence must be taken into account, so that if it becomes unsatisfiable after one narrowing step, it will be discarded.

In this way, we eliminate redundancy as well as branches of the reachability tree, which will be less and less wide than the tree resulting from using standard narrowing. In some cases, we will even get infinite reachability trees to become finite, ensuring termination.

The key completeness property about this relation is the following.

Lemma 1 (Lifting Lemma). *Given $\langle t, \Pi, \varphi\rangle$, a $\overrightarrow{E}, B$-normalized substitution θ, and terms $u, v \in \mathcal{C}^{\circ}_{\Sigma/E,B}(\mathcal{X})$ such that $u = (t\theta)\downarrow_{\overrightarrow{E},B}$, $T \models \varphi\theta$, and $\Pi\theta$ are $\overrightarrow{E}, B$-normalized and $u \to_{R/E,B} v$, there is a canonical narrowing step with irreducibility and SMT constraints*

$$\langle t, \Pi, \varphi\rangle \rightsquigarrow_{\alpha,R/E,B} \langle r\alpha, \Pi\alpha \cup \{(l\alpha)\downarrow_{\overrightarrow{E},B}\}, \varphi'\rangle$$

and a $\overrightarrow{E}, B$-normalized substitution γ such that

$$\begin{array}{ccc} \langle t, \Pi, \varphi\rangle & \rightsquigarrow_{\alpha,R/E,B} & \langle r\alpha, \Pi\alpha \cup \{(l\alpha)\downarrow_{\overrightarrow{E},B}\}, \varphi'\rangle \\ \downarrow_{\theta} & & \downarrow_{\gamma} \\ \lceil\langle t, \Pi, \varphi\rangle\rceil_{\overrightarrow{E},B} & \to_{R/E,B} & \lceil\langle r\alpha, \Pi\alpha \cup \{(l\alpha)\downarrow_{\overrightarrow{E},B}\}, \varphi'\rangle\rceil_{\overrightarrow{E},B} \end{array}$$

(i) $\theta =_B (\alpha\gamma)|_{Var(\langle t,\Pi,\varphi\rangle)}$,
(ii) $(r\alpha\gamma)\downarrow_{\overrightarrow{E},B} =_B v$,
(iii) $\Pi\alpha\gamma \cup \{((l\alpha)\downarrow_{\overrightarrow{E},B})\gamma\}$ *are* $\overrightarrow{E}, B$*-normalized,*
(iii) $T \models \varphi'\gamma$.

Note that this shows that $v \in \lceil\langle r\alpha, \Pi\alpha \cup \{(l\alpha)\downarrow_{\overrightarrow{E},B}\}, \varphi'\rangle\rceil_{\overrightarrow{E},B}$.

4 Implementation

To implement SMT constraint handling in the narrowing algorithm, we have used our implementation of standard/canonical narrowing [14] as a starting point. To do this, we use the features of the Maude meta-level, thus creating an extension of the previous meta-level command.

4.1 Our Previous Narrowing Command

The meta-level command we use as a starting point already allows us to choose between several narrowing algorithms to use. First of all, it allows to invoke the standard narrowing algorithm, with a behavior similar to the standard narrowing built-in in Maude. It also allows the canonical narrowing algorithm [14] to be invoked, in which irreducibility constraints are used to reduce the width of the computed reachability tree. To control the algorithm used along with other parameters, such as the maximum depth of the tree or the maximum number of solutions to search for, the command uses ten arguments:

```
narrowing(Module, Term, SearchArrow, Term, AlgorithmOptionSet, VariantOptionSet, TermList, Qid,
          Bound, Bound)
```

In the implementation of that command, we already prepared an adequate infrastructure to allow future extensions. Several data structures and substructures were defined to represent the reachability tree, its nodes, and the solutions found. Additionally, we divided the implementation into three main parts, which correspond to the main steps of the algorithm at a theoretical level: (i) the generation of nodes (terms) in the reachability tree, (ii) the attempt to unify each new term with the target term, and (iii) the computation of solutions in case the unification is successful. Those main parts are further broken down into highly distinguishable subparts, making it easy to make extensions or modifications to some parts without having to change the rest of the implementation.

4.2 Using Conditional Rules in Narrowing

To manage SMT constraints, our approach has been to use Maude's conditional rules to add them as a condition in each of the narrowing steps. The problem that arises is that the Maude narrowing mechanisms are not capable of processing the conditional rules. The way to fix this is to transform those conditional rules into normal rules, in which the new left-hand side of the new rules will contain both the left-hand side of the conditional rules and the SMT constraints. An operator should separate both parts, so that later the original term can be distinguished from the SMT restrictions.

We have implemented a module that is responsible for carrying out the process of transformation of conditional rules. This module defines two operators:

```
op transformMod : Module -> Module .
op transformRls : RuleSet -> RuleSet .
```

The first receives a module, theory, module with strategy or theory with strategy. In either case, a new operator is added to the set of operators of the module or theory, which will be used to separate the terms from the SMT constraints in the transformed rules. It is also necessary to add the import of the Maude `META-TERM` module to the converted module, so that it is capable of processing the addition of this new operator. Finally, this operator calls the other defined operator, using as an argument the set of rules of the module to be transformed. For example, the equation used to transform a module without strategy would be the following:

```
eq transformMod(mod ModId is Imports sorts Sorts . Subsorts Ops Membs Eqs Rls1 endm)
    = mod ModId is Imports (protecting 'META-TERM .)
      sorts Sorts . Subsorts
      (Ops (op '_>>_ : 'Boolean 'State -> 'State [ctor poly (0 2)] .))
      Membs Eqs transformRls(Rls1) endm .
```

The second operator, therefore, receives a set of rules, and is in charge of iterate through it looking for conditional rules. Each time a conditional rule is found, it is transformed into a new unconditional rule, in which the condition is added to the left-hand side using the `>>` operator defined above. The equations used to do this are as follows:

```
eq transformRls(Rls1 (crl Lhs => Rhs if (SmtConst = BooleanValue) [Attrs].) Rls2)
    = transformRls(Rls1 Rls2) (rl Lhs => '_>>_[SmtConst,Rhs] [Attrs narrowing] .) .
eq transformRls(Rls1) = Rls1 [owise] .
```

Thus, if we have a conditional rule of the form `crl Lhs => Rhs if (SmtConst = BooleanValue) [Attrs]`, it will be automatically transformed into an unconditional rule of the form `rl Lhs => (SmtConst >> Rhs) [Attrs narrowing]`, where `Lhs` and `Rhs` are variables of `Universal` type (that is, they can be instantiated as any sort), `SmtConst` is a variable that represents the SMT constraints, and `BooleanValue` is a `Boolean` variable expected to be true, used only to be able to encode SMT constraints in the conditions of the rules. The new form of the rule after transforming it will allow us later to make the `>>` operator disappear and separate the term from the SMT constraints. This is explained in detail in the following section.

4.3 Extension to Handle SMT Constraints

Once we have prepared the module transformation to convert all the conditional rules into unconditional ones, we can extend the previous command so that it processes the SMT terms that will be generated with the new rules. This extension has been done without making changes at the user level, except for the addition of possible values to one of the existing arguments, as well as a new argument that allows to indicate initial SMT constraints:

```
narrowing(Module, Term, SearchArrow, Term, AlgorithmOptionSet, VariantOptionSet, TermList,
          Term, Qid, Bound, Bound)
```

Until now, the fifth argument, of type `AlgorithmOptionSet`, only accepted the `standard` and `canonical` values, used to indicate the type of narrowing

algorithm to use. Now, it also accepts combinations of those two values with the `smt` and `noCheck` values, although the second is a limitation of the first, so it cannot appear without it. By using the `smt` value, the transformation of the conditional rules will be performed in the module used as rewrite theory if necessary. Subsequently, the SMT constraints will be processed during the execution of the algorithm to check if they are satisfiable at each node of the reachability tree. If it is also accompanied by the value `noCheck`, only the transformation of the rules will be carried out, ignoring the satisfiability of the SMT constraints.

The most relevant changes to the algorithm occur before trying to unify the term of a new generated node with the target term, since the satisfiability of the SMT constraints for that node will have to be checked first. Until reaching that step, not many modifications are needed, since the narrowing steps will be given using the rules in a usual way, because the transformation of conditional rules will have been previously carried out just at the beginning of the algorithm, if the user indicates that SMT constraints are being used. Furthermore, we need to modify the previously used data structures. Now the main structure must save the initial SMT constraints indicated by the user. It will also be necessary for each of the nodes to contain a list of the SMT constraints carried so far. We have stored that list at each node in a {`Term, Bool`} pair, where the second value of the pair indicates the satisfiability of the constraints found in the first value. Two new operators are introduced in the algorithm that run after the generation of a new node and renaming of its variables, although they will only be used if the user indicates that SMT restrictions must be processed:

```
op evaluateSMT : UserArguments TreeInfo SolutionList -> NarrowingInfo .
op checkSat : UserArguments TreeInfo SolutionList -> NarrowingInfo .
```

The `evaluateSMT` operator performs the separation of the SMT constraints from the new term generated with one of the transformed rules. In turn, it joins these restrictions with the list of restrictions carried so far, which will come from the predecessor nodes to the current one and from the initial restrictions indicated by the user. Additionally, it launches to evaluate all those restrictions, to know if they are satisfiable or not. To do this, we rely on Maude's SMT interface, which is available in the meta-level. Specifically, we use the `metaCheck` [5, §16], which receives the module to use and the term to evaluate, returning a value of type `Bool`. If the result is `true`, the constraints are satisfiable. Otherwise, `false` is returned:

```
op metaCheck : Module Term ~> Bool [special (...)] .
```

Note that in case the user has indicated, in addition to the `smt` value as an argument, the `noCheck` value, the `evaluateSMT` operator will only separate the SMT constraints from the term, ignoring the rest of the process, since we are not interested in checking the satisfiability, but in being able to process the constraints of the initial conditional rules.

The `checkSat` operator is responsible for processing the result obtained when executing the `metaCheck` function. If the restrictions are satisfiable, the next execution step should be the attempt to unify the term of the node with the

objective term, to check if it corresponds to one or more solutions of the reachability problem. If the constraints are not satisfiable, then it will not make sense to perform the unification step, since we will not consider the term of the node as valid. We therefore return to the step of generating new nodes, marking the current node as invalid, so that it is not taken into account later, since we do not want to generate the possible child nodes of this node either.

4.4 Variable Consistency

As we explained in our previous work [14] on which we based this algorithm, the way Maude generates the fresh variables may lead to clashes. For this reason, the fresh variables that are generated in each narrowing step must be renamed using an internal counter, and using the $ symbol as an identifier. Since the variables in the SMT constraints are related to those used in the terms, as well as to the variables in the previously processed SMT constraints, there is a consistency problem with this renaming. That is why in each narrowing step, we now have to apply variable substitutions to the SMT constraints so that there is no such loss of consistency. Specifically, at each narrowing step, the computed substitution that must be applied to the term of the previous node to take that step must be applied to the new node's SMT constraint. The substitution must also be applied to the SMT constraint list carried along the node branch. In turn, this list will already come with the variables renamed in the previous steps, so consistency builds up. Note that the initial SMT restrictions indicated by the user will also have to be renamed. This is not a problem, since these constraints are also automatically added to the list of constraints of each node, so it can be renamed at the same time as the rest.

5 Experiments

For the experiments, we have considered the Brands and Chaum protocol of Example 2 in two forms: its version with only time, published in [1], and its version with time and space, published in [2]. In both, the use of SMT restrictions is necessary, which in our case are codified with conditional rules. As explain in Sect. 4, these conditional rules will be processed to transform them into unconditional rules, in order to correctly obtain the SMT constraints at each narrowing step.

All the files used to define the new narrowing algorithm, as well as the experiments that we will see next and their results, can be found at the following link: https://github.com/ralorueda/smt-narrowing.

5.1 Handling SMT Constraints

We rely on the generic rewrite theory for protocol specification, inspired on the strand spaces [12] used by Maude-NPA [8], used in our previous work on canonical narrowing [14], but with some modifications that adapt it to include

SMT constraints on the real numbers, inspired on the constraints used in [1,2]. It is a module that allows us to specify a state, made up of sets of strands and the intruder knowledge, which represents the communication channel. With it we can represent the protocols in a generic way, adding the corresponding equational theories for each of them. Later, when coding the narrowing calls, we will specify the exact strands of each protocol.

In the original module, we had two transition rules. One of them processes the sent messages, and the other the received messages:

```
var IK : IntruderKnowledge .   var SS : StrandSet .   var M : Msg .   vars L1 L2 : SMsgList .

rl [receive-msg] : { (SS & [ ( L1 , -(M)) | L2 ])  { (inI(M) , IK) } } =>
                   { (SS & [ L1 | (-(M) , L2) ])  { (inI(M) , IK) } } [narrowing] .

rl [send-msg] : { (SS & [ (L1 , +(M)) | L2 ])  { (inI(M) , IK) } } =>
                   { (SS & [ L1 | (+(M) , L2) ])  { (nI(M) , IK) } } [narrowing] .
```

It can be seen in each of them how, for each set of strands, represented in square brackets, there is a list to the left of the operator | and one to the right. The first contains the messages to be processed, while the second contains the processed messages. At each transition, a message (sent or received) is taken from the end of the list of messages to be processed and moved to the list of processed messages. In the event that it is a sent message, the correspondence of that message will also be modified in the communication channel or intruder knowledge.

To adapt the module to protocols using non-linear arithmetic constraints on the real numbers via satisfiability, we add a conditional rule that is responsible for processing a new type of data that can appear in the strands sets: constraints. Specifically in our case, SMT constraints (type `Boolean`), which will be represented in the channel between the messages with the operator {_}. We will therefore now have three rules. One of them is responsible for processing the messages sent, another the messages received, and another the restrictions that occur at any given time:

```
var IK : IntruderKnowledge .   var SS : StrandSet .   var SSR : StrandSetR .
var SSN : StrandSetN .   var M : Msg .   vars LeE2 : SMsgList-eE .
var LREe1 : SMsgListR-Ee .

crl [check-contraint] : { (SSR & [ LREe1 , {B:Boolean} | LeE2 ])  { IK } } =>
                   { (SSR & [ LREe1 | {B:Boolean} , LeE2 ])  { IK } }
                      if B:Boolean = true [nonexec] .
rl [receive-msg] : { (SSN & [ LREe1 , -(M) | LeE2 ])  { (inI(M) , IK) } } =>
                   { (SSN & [ LREe1 | -(M) , LeE2 ])  { (inI(M) , IK) } } [narrowing] .
rl [send-msg] : { (SS & [ LREe1 , +(M) | LeE2 ])  { (inI(M) , IK) } } =>
                   { (SS & [ LREe1 | +(M) , LeE2 ])  { (nI(M) , IK) } } [narrowing] .
```

Note that in this case we use variables from different sorts, `SMsgListR-Ee` and `SMsgListR-eE`, rather than the ones we used in [14]. This is because we have created a rule hierarchy, mimicking some optimizations of the Maude-NPA [9], in such a way that a more defined processing order is followed, significantly reducing the computation time in the experiments. In this way, whenever there is a constraint at the end of the list of messages to be processed in a strand set, it will be processed first. If this is not the case, it will check if there is any received message at the end of the list of messages to be processed in a strand set, and

will be processed. If neither of these two cases occurs, then a sent message will be processed.

5.2 Brands and Chaum with Time

The previous module allows us, in a generic way, to specify protocols that contain SMT restrictions. To this must be added the specific equational theories of each protocol. In our case, the first protocol used is Brands and Cham with time [1], which can be seen as a simplified version of the protocol seen in Example 1, but does not take into account the coordinates of the messages. Two cryptographic primitives are combined: exclusive-or over a set of nonces and a commitment scheme. Exclusive-or is defined with the following properties:

```
sort NNSet .
subsorts Nonce Secret < NNSet .

op null : -> NNSet .
op _*_ : NNSet NNSet -> NNSet [assoc comm] .
vars X Y : [NNSet] .

eq [idem] :     X * X = null    [variant] .
eq [idem-Coh] : X * X * Y = Y   [variant] .
eq [id] :       X * null = X    [variant] .
```

The commitment scheme allows a participant to commit to a chosen hidden value at an early protocol stage and reveal it later. It is defined with the following properties:

```
op commit : Nonce Secret -> NTMsg .
op open : Nonce Secret NTMsg -> [Boolean] .
eq open(N1:Nonce,Sr:Secret,commit(N1:Nonce,Sr:Secret)) = true [variant] .
```

The `open` function is defined only for the successful case. This implies the use of the kind `[Boolean]` rather than the sort `Boolean`. We also use additional operators for this protocol, which allow us to define signing, message concatenation, and the creation of nonces and secrets.

```
sorts Msg NTMsg TMsg .
sorts Name Honest Intruder Fresh Secret Nonce .
subsorts NNSet < NTMsg < Msg .
subsorts Nonce Secret < NNSet .
subsort Name < Msg .
subsort Honest Intruder < Name .

ops a b : -> Honest .
op i : -> Intruder .
ops ra1 rb1 rb2 : -> Fresh .
op n : Name Fresh -> Nonce .
op s : Name Fresh -> Secret .
op sign : Name NTMsg -> NTMsg .
op _;_ : NTMsg  NTMsg  -> NTMsg [gather (e E)] .
```

Additionally, we add several operators that will allow us to add metadata to the messages. In them, the sending and receiving times of the messages will be saved, as well as the identifier of the sender and the receiver.

```
sorts TimeInfo NameTime NameTimeSet .
subsort NameTime < NameTimeSet .
subsort TMsg < Msg .

op _@_ : NTMsg TimeInfo -> TMsg .
op _:_ : Name Real -> NameTime .
op mt : -> NameTimeSet .
op _#_ : NameTimeSet NameTimeSet -> NameTimeSet [assoc comm id: mt] .
op _->_ : NameTime NameTimeSet -> TimeInfo .
```

Note that times will be represented as real numbers, one of the data types manageable by Maude's SMT interface. The distance between two participants A and B is represented by a variable `dab:Real`.

The module defined with the previous sorts, operators and rules allows us to code the strands of the Brands and Chaum protocol of Example 1 only with time. This will be done in the call to the narrowing algorithm, with an initial state and a target state. In the initial state, the strand sets will contain a list of messages and constraints to be processed and a list of messages and constraints processed, which will be empty. In the target state, the lists will have been inverted, so that all the messages and restrictions to be processed become processed. Consider, for example, the strands of a prover and a verifier in a regular execution of the Brands and Chaum protocol with time. With our syntax, they would be specified in the initial state as follows:

```
--- Alice, verifier
([nilEe,
  -(Commit:NTMsg                                @ b : t1:Real -> a : t1':Real),
        {(t1':Real === t1:Real + dab:Real) and dab:Real > 0/1},
  +(n(a,ra1)                                    @ a : t2:Real -> b : t2':Real),
  -((n(a,ra1) * NB:Nonce)                       @ b : t3:Real -> a : t3':Real),
        {(t3':Real === t3:Real + dab:Real) and dab:Real > 0/1 and t3:Real >= t2':Real},
        {(t3':Real - t2':Real) <= (2/1 * dab:Real) and dab:Real > 0/1},
  -(SB:Secret                                   @ b : t4:Real -> a : t4':Real),
        {open(NB:Nonce,SB:Secret,Commit:NTMsg)},
        {(t4':Real === t4:Real + dab:Real) and dab:Real > 0/1 and t4:Real >= t3':Real},
  -(sign(b,(n(a,ra1) * NB:Nonce) ; n(a,ra1))    @ b : t5:Real -> a : t5':Real),
        {(t5':Real === t5:Real + dab:Real) and dab:Real > 0/1 and t5:Real >= t4':Real}
| nileE]
&
--- Bob, prover
[nilEe,
  +(commit(n(b,rb1),s(b,rb2))                   @ b : t1:Real -> a : t1':Real),
  -(NA:Nonce                                    @ a : t2:Real -> b : t2':Real),
        {(t2':Real === t2:Real + dab:Real) and dab:Real > 0/1 and t2:Real >= t1':Real},
  +((NA:Nonce * n(b,rb1))                       @ b : t3:Real -> a : t3':Real),
  +(s(b,rb2)                                    @ b : t4:Real -> a : t4':Real),
  +(sign(b,(NA:Nonce * n(b,rb1)) ; NA:Nonce)    @ b : t5:Real -> a : t5':Real)
| nileE])
```

We can see how the prover, Bob, will first send a commit to the verifier. Afterwards, the verifier, Alice, sends her nonce to the prover. Subsequently, the prover will send the XOR of his nonce with the received one, and then sends the secret. The verifier will open it to confirm everything is okay. Finally, the prover will send the signs messages. An `@` operator appears in each message, after which the sending and receiving times of the message are saved, as well as the identifier of the sender and receiver. We can also see how SMT constraints are introduced after each received message. In them, conditions to be met are specified regarding

the delivery and reception times. Conditions to satisfy relative to distances are also specified. For example, in the SMT constraint that is introduced on the strands of the prover, it is specified that the arrival time of the received message must be equal to its departure time plus the distance between the prover and the verifier. It is also specified that this distance must be greater than zero, and that the sending time of the message must be equal to or greater than the time in which the previous message was received.

Using this syntax and coding methodology, we have defined three experiments in which we test a regular execution of the protocol, a mafia-like attack pattern, and a hijacking-like attack pattern. In regular execution, we get a solution, which is expected, since if the protocol is well defined, this execution should be possible. In the case of the mafia attack, a priori, a solution is also found, which translates into a possible vulnerability. However, adding the triangle inequality $(d(a,i) + d(b,i)) > d$ as the initial constraint, together with the constraint $d(V,P) > d > 0$, no solution is found. This is because, for consistency to exist in this execution, it is necessary that $2 * d(V,I) + 2 * d(P,I) \leq 2 * d$. As mentioned in Sect. 4, the initial SMT constraints can be written in one of the arguments of the narrowing command. However, it is possible to perform a hijacking attack, and that is why by specifying this pattern in one of the experiments, a solution is found. The attack occurs when an intruder located outside the neighborhood of the verifier (i.e., $d(V,I) > d$) succeeds in convincing the verifier that he is inside the neighborhood by exploiting the presence of an honest prover in the neighborhood (i.e., $d(V,P) \leq d$).

5.3 Brands and Chaum with Time and Space

The second protocol that we have used for the experiments is an extension of the previous one: Brands and Chaum with time and space, detailed at a theoretical level in Example 2. In this case, the coordinates related to the sending and receiving of each message appear in the metadata of the messages and in the restrictions, that is, the coordinates of the participants. To be able to write this, a slight modification of the previous protocol specification is enough, as well as the addition of a new operator:

```
sort CoordNameTime .
op _:_,_,_,_ : Name Real Real Real Real -> CoordNameTime .
op _->_ : CoordNameTime NameTimeSet -> TimeInfo .
```

Once the modification is done, it is possible to encode the new strands. For example, the strands for a verifier and a prover in a regular execution of the protocol would now be as follows:

```
--- Alice, verifier
[nilEe,
  -(Commit:NTMsg
        @ b : x1:Real,y1:Real,z1:Real,t1:Real -> a : t2:Real),
            {(t2:Real === t1:Real + dab1:Real) and (dab1:Real > 0/1) and
            ((dab1:Real * dab1:Real) === (((x1:Real - ax:Real) * (x1:Real - ax:Real)) +
            ((y1:Real - ay:Real) * (y1:Real - ay:Real))) +
            ((z1:Real - az:Real) * (z1:Real - az:Real)))},
```

```
  +(n(a,ra1)
        @ a : ax:Real,ay:Real,az:Real,t2:Real -> b : t3:Real),
  -((n(a,ra1) * NB:Nonce)
        @ b : x3:Real,y3:Real,z3:Real,t3:Real -> a : t4:Real),
            {(t4:Real === t3:Real + dab3:Real) and (dab3:Real > 0/1) and
            ((dab3:Real * dab3:Real) === (((x3:Real - ax:Real) * (x3:Real - ax:Real)) +
            ((y3:Real - ay:Real) * (y3:Real - ay:Real))) +
            ((z3:Real - az:Real) * (z3:Real - az:Real)))},
            {((t4:Real - t2:Real) <= (2/1 * d:Real)) and (d:Real > 0/1)},
  -(SB:Secret
        @ b : x4:Real,y4:Real,z4:Real,t5:Real -> a : t6:Real),
            {open(NB:Nonce,SB:Secret,Commit:NTMsg)},
            {(t6:Real === t5:Real + dab4:Real) and (dab4:Real > 0/1) and
            ((dab4:Real * dab4:Real) === (((x4:Real - ax:Real) * (x4:Real - ax:Real)) +
            ((y4:Real - ay:Real) * (y4:Real - ay:Real))) +
            ((z4:Real - az:Real) * (z4:Real - az:Real)))},
  -(sign(b,(n(a,ra1) * NB:Nonce) ; n(a,ra1))
        @ b : x5:Real,y5:Real,z5:Real,t7:Real -> a : t8:Real),
            {(t8:Real ===  t7:Real + dab5:Real) and (dab5:Real > 0/1) and
            ((dab5:Real * dab5:Real) === (((x5:Real - ax:Real) * (x5:Real - ax:Real)) +
            ((y5:Real - ay:Real) * (y5:Real - ay:Real))) +
            ((z5:Real - az:Real) * (z5:Real - az:Real)))}
| nileE]
&
--- Bob, prover
[nilEe,
  +(commit(n(b,rb1),s(b,rb2))
        @ b : bx:Real,by:Real,bz:Real,t1:Real -> a : t2:Real),
  -(NA:Nonce
        @ a : x2:Real,y2:Real,z2:Real,t2:Real -> b : t3:Real),
            {(t3:Real === t2:Real + dab2:Real) and (dab2:Real > 0/1) and
            ((dab2:Real * dab2:Real) === (((x2:Real - bx:Real) * (x2:Real - bx:Real)) +
            ((y2:Real - by:Real) * (y2:Real - by:Real))) +
            ((z2:Real - bz:Real) * (z2:Real - bz:Real)))},
  +((NA:Nonce * n(b,rb1))
        @ b : bx:Real,by:Real,bz:Real,t3:Real -> a : t4:Real),
  +(s(b,rb2)
        @ b : bx:Real,by:Real,bz:Real,t3:Real -> a : t6:Real),
  +(sign(b,(NA:Nonce * n(b,rb1)) ; NA:Nonce)
        @ b : bx:Real,by:Real,bz:Real,t3:Real -> a : t8:Real)
| nileE]
```

The exchange of messages is very similar to what we have seen before, but in this case the metadata is somewhat more complex, since the sending coordinates are attached to each sending time. In addition, the restrictions are also complicated, since in this case it will also be necessary to verify that the conditions required for those coordinates are satisfied at each moment. In fact, since the new constraints are non-linear arithmetic, Maude's SMT is not capable of processing them. In order to correctly execute the traces related to this protocol, we have used a version of Maude called Maude-NRA, which provides an SMT solver (CVC4) that is capable of processing this type of arithmetic.

Once more, we have performed experiments for this protocol with a regular execution, a mafia-like attack pattern, and a hijacking-like attack pattern. The results are similar to the previous ones, although more complex. Regular execution returns a solution, since it is possible to do it without problems. The hijacking attack is again possible as well, so a solution is again returned. Regarding the mafia attack, the same thing happens: a priori it is possible, but by adding the initial SMT restrictions necessary for the trace to be consistent,

the attack is impossible. These restrictions are the same as before, but in this case some relative to coordinates are also added.

6 Conclusions and Future Work

The canonical narrowing strategy with irreducibility and SMT constraints opens the door to the use of narrowing to analyze protocols that use laws of physics, such as the Brands and Chaum protocol. It is a greatly generic methodology of symbolic reachability analysis that manages to prove the existence of traces of a protocol, giving greater flexibility when defining and specifying them. In this article we have presented an implementation of canonical narrowing capable of handling SMT constraints. This allows us to carry out symbolic analysis of two versions of the Brands and Chaum protocol. Maude-NPA already handled such protocols, as shown in [1,2], but in an ad-hoc way without the canonical narrowing presented here. We now have a new algorithm with a powerful theoretical framework behind it, which can be useful to both Maude-NPA and other symbolic protocol analysis tools. As future work, we expect to expand this canonical narrowing to more general cases, clearly increasing its power for protocol analysis.

References

1. Aparicio-Sánchez, D., Escobar, S., Meadows, C., Meseguer, J., Sapiña, J.: Protocol analysis with time. In: Bhargavan, K., Oswald, E., Prabhakaran, M. (eds.) INDOCRYPT 2020. LNCS, vol. 12578, pp. 128–150. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-65277-7_7
2. Aparicio-Sánchez, D., Escobar, S., Meadows, C., Meseguer, J., Sapiña, J.: Protocol analysis with time and space. In: Dougherty, D., Meseguer, J., Mödersheim, S.A., Rowe, P. (eds.) Protocols, Strands, and Logic. LNCS, vol. 13066, pp. 22–49. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-91631-2_2
3. Brands, S., Chaum, D.: Distance-bounding protocols. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 344–359. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48285-7_30
4. Chadha, R., Cheval, V., Ciobâcă, Ş., Kremer, S.: Automated verification of equivalence properties of cryptographic protocols. ACM Trans. Comput. Log. **17**(4), 23:1–23:32 (2016)
5. Clavel, M., et al: Maude Manual (Version 3.2.1). Technical report, SRI International Computer Science Laboratory (2022). http://maude.cs.illinois.edu
6. Comon-Lundh, H., Delaune, S.: The finite variant property: how to get rid of some algebraic properties. In: Giesl, J. (ed.) RTA 2005. LNCS, vol. 3467, pp. 294–307. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-32033-3_22
7. Erbatur, S., et al.: Asymmetric unification: a new unification paradigm for cryptographic protocol analysis. In: Bonacina, M.P. (ed.) CADE 2013. LNCS (LNAI), vol. 7898, pp. 231–248. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38574-2_16

8. Escobar, S., Meadows, C., Meseguer, J.: Maude-NPA: cryptographic protocol analysis modulo equational properties. In: Aldini, A., Barthe, G., Gorrieri, R. (eds.) FOSAD 2007-2009. LNCS, vol. 5705, pp. 1–50. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03829-7_1
9. Escobar, S., Meadows, C.A., Meseguer, J., Santiago, S.: State space reduction in the Maude-NRL protocol analyzer. Inf. Comput. **238**, 157–186 (2014)
10. Escobar, S., Meseguer, J.: Canonical narrowing with irreducibility constraints as a symbolic protocol analysis method. In: Guttman, J.D., Landwehr, C.E., Meseguer, J., Pavlovic, D. (eds.) Foundations of Security, Protocols, and Equational Reasoning. LNCS, vol. 11565, pp. 15–38. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-19052-1_4
11. Escobar, S., Sasse, R., Meseguer, J.: Folding variant narrowing and optimal variant termination. J. Log. Algebr. Program. **81**(7–8), 898–928 (2012)
12. Thayer Fabrega, F.J., Herzog, J., Guttman, J.: Strand spaces: what makes a security protocol correct? J. Comput. Secur. **7**, 191–230 (1999)
13. Jouannaud, J.-P., Kirchner, H.: Completion of a set of rules modulo a set of equations. SIAM J. Comput. **15**(4), 1155–1194 (1986)
14. López-Rueda, R., Escobar, S., Meseguer, J.: An efficient canonical narrowing implementation for protocol analysis. In: Bae, K. (ed.) WRLA 2022. LNCS, vol. 13252, pp. 151–170. Springer, Cham (2022). Held as a Satellite Event of ETAPS, Munich, Germany, 2–3 April 2022, Proceedings
15. Meier, S., Schmidt, B., Cremers, C., Basin, D.: The TAMARIN prover for the symbolic analysis of security protocols. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 696–701. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_48
16. Meseguer, J.: Conditioned rewriting logic as a united model of concurrency. Theor. Comput. Sci. **96**(1), 73–155 (1992)
17. Meseguer, J.: Membership algebra as a logical framework for equational specification. In: Presicce, F.P. (ed.) WADT 1997. LNCS, vol. 1376, pp. 18–61. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-64299-4_26
18. Meseguer, J.: Strict coherence of conditional rewriting modulo axioms. Theor. Comput. Sci. **672**, 1–35 (2017)
19. Meseguer, J.: Generalized rewrite theories, coherence completion, and symbolic methods. J. Log. Algebraic Methods Program. **110**, 100483 (2020)
20. Meseguer, J., Thati, P.: Symbolic reachability analysis using narrowing and its application to verification of cryptographic protocols. High.-Order Symb. Comput. **20**(1–2), 123–160 (2007)
21. TeReSe (ed.): Term Rewriting Systems. Cambridge University Press, Cambridge (2003)