Narrowing-based Optimization of Rewrite Theories*

María Alpuente Santiago Escobar Julia Sapiña
VRAIN, Universitat Politècnica de València
Valencia, Spain
{alpuente,sescobar,jsapina}@upv.es

Demis Ballis
DMIF, University of Udine
Udine, Italy
demis.ballis@uniud.it

Partial evaluation has been never investigated in the context of rewrite theories that allow concurrent systems to be specified by means of rules, with an underlying equational theory being used to model system states as terms of an algebraic data type. In this paper, we develop a symbolic, narrowing-driven partial evaluation framework for rewrite theories that supports sorts, subsort overloading, rules, equations, and algebraic axioms. Our partial evaluation scheme allows a rewrite theory to be optimized by specializing the plugged equational theory with respect to the rewrite rules that define the system dynamics. This can be particularly useful for automatically optimizing rewrite theories that contain overly general equational theories which perform unnecessary computations involving matching modulo axioms, because some of the axioms may be blown away after the transformation. The specialization is done by using appropriate unfolding and abstraction algorithms that achieve significant specialization while ensuring the correctness and termination of the specialization. Our preliminary results demonstrate that our transformation can speed up a number of benchmarks that are difficult to optimize otherwise.

1 Introduction

Rewriting Logic (RWL) is a logic of change that extends order-sorted equational logic by adding rewrite rules that are used to describe non-deterministic transitions of concurrent systems. Rewriting Logic is efficiently implemented in the high-performance system Maude [6]. Roughly speaking, a rewrite theory seamlessly combines a *term rewriting system* (TRS), which specifies the system dynamics, with an *equational theory* that defines the static structure of the system states. The equational theory may contain equations and axioms (i.e., distinguished equations that specify algebraic laws such as commutativity, associativity, and unity for some theory operators) so that rewrite steps are performed *modulo* the equations and axioms.

Partial evaluation (PE) is a program optimization technique (also known as program specialization) that, given a program and some of its input data, produces a residual or specialized program. Running the residual program on the remaining data is generally faster and yields the same result as running the original program on all of its input data [12]. PE has been widely applied to a variety of programs, including functional programs (FP) [12] and logic programs (LP) [13], where it is usually called *partial deduction* (PD). The *Equational Narrowing-driven Partial Evaluation* (EqNPE) scheme of [1] extends PD to the specialization of order-sorted equational theories with respect to a set of input terms. The input equational theory $(\Sigma, E \uplus B)$ consists of a set E of convergent equations (that in the order-sorted setting means they are confluent and terminating, among other requirements) that are implicitly oriented from left to right as rewrite rules (and operationally used as simplification rules), and a set B of commonly

*This work has been partially supported by the EU (FEDER) and the Spanish MCIU under grant RTI2018-094403-B-C32, and by Generalitat Valenciana under grant PROMETEO/2019/098. Julia Sapiña has been supported by the Generalitat Valenciana APOSTD/2019/127 grant.

occurring axioms such associativity, commutativity, and identity that are essentially used for pattern matching modulo axioms. Thanks to the use of (a form of) *narrowing*, the symbolic mechanism that extends term rewriting by replacing pattern matching with unification [17], the achieved transformation is strictly more powerful than the PE of both logic programs and functional programs [4] In EqNPE, the key ingredients of PD are generalized to an order-sorted typed setting modulo axioms by providing: 1) a narrowing-based, tree *unfolding operator* for ensuring correction of the transformation; 2) *order-sorted equational homeomorphic embedding* for local termination (i.e., finiteness of unfolding); 3) *order-sorted (recursive) equational closedness* (that ensures that all possible calls that may arise during the execution of the residual program are covered by the specialization) for completeness; and 4) term abstraction (based on *order-sorted equational least general* generalization) for global termination of the whole specialization process.

In the following we consider a rewrite theory $\mathcal{R} = (\Sigma, E \uplus B, R)$ that extends the order-sorted equational theory $\mathcal{E} = (\Sigma, E \uplus B)$ with a set R of rewrite rules that specify concurrent system transitions. Rewrite theories can not only be executed by equational rewriting in Maude but also *symbolically* executed by narrowing at *two levels*: (i) narrowing with the (typically non-confluent and non-terminating) rules of R modulo $(E \uplus B)$, and (ii) narrowing with oriented equations $\vec{E}$ (the explicitly oriented version of the equations in E) modulo the axioms B. They both have practical applications: (i) narrowing with R modulo $(E \uplus B)$ is useful for solving *reachability goals* [15] and *logical model checking* [9], and (ii) narrowing with $\vec{E}$ modulo B (or $(\vec{E}, B)$-narrowing) is useful for equational unification and variant computation [10]. Both levels of narrowing should meet some conditions: (i) narrowing with R modulo $(E \uplus B)$ is performed in a "topmost" way (i.e., the rules in R rewrite the global system state) and there must be a finitary unification algorithm for $(E \uplus B)$, and (ii) narrowing with $\vec{E}$ modulo B requires that B is a theory with a finitary unification algorithm and that $\vec{E}$ is convergent. When $(\Sigma, E \uplus B)$ additionally has the property that a finite complete set of most general $(\vec{E},B)$-*variants*[1] exists for each term, known as the *finite variant property* (FVP), $\mathcal{E}$-unification is finitary [7], and *topmost* narrowing with R modulo the equations and axioms can be effectively performed.

For $(\vec{E},B)$-variant computation and (variant-based) $\mathcal{E}$-unification, the *folding variant narrowing (FVN) strategy* of [10] is used. The main idea of folding variant narrowing is to "fold"[2] , by subsumption modulo B, the $(\vec{E}, B)$-narrowing tree that can in practice result in a finite, directed acyclic narrowing graph that symbolically and concisely summarizes the (generally infinite) narrowing search space of the theory. Nevertheless, finiteness of folding variant narrowing trees is only guaranteed for equational theories that satisfy the finite variant property.

Because both reachability goals and logical model checking generally require the whole search space of rewrite theories to be analyzed (i.e. all system states and transitions), the opportunities for optimizing rewrite theories by partial evaluation may appear to be scarce. Actually, partial evaluation typically removes some computation states by performing as much program computation as possible hence contracting the search space because some transitions are removed. The key idea in this paper for the specialization of a rewrite theory $\mathcal{R}$ is to partially evaluate the underlying equational theory $\mathcal{E}$ with respect to the function calls in the rules of $\mathcal{R}$

[1] A variant of a term t consists of a pair (t', σ), where t' is the $(\vec{E},B)$-irreducible form of $t\sigma$ for a substitution σ.

[2] The notion of folding in folding variant narrowing is essentially a *subsumption* notion applied to some leaves of the narrowing tree so that less general leaves are subsumed (folded into) most general ones. Therefore, this notion is quite different from the classical folding operation of Burstall and Darlington's fold/unfold transformation scheme [5], where unfolding is essentially the replacement of a call by its body, with appropriate substitutions, and folding is the inverse transformation, i.e., the replacement of some piece of code by an equivalent function call.

in such a way that $\mathcal{E}$ gets rid of any unneeded overgenerality. By this means, only the functional computations in $\mathcal{E}$ are compressed by partial evaluation, while keeping the concurrent computations with the rules of R. Depending on the properties of both, $\mathcal{E}$ and $\mathcal{R}$, the right unfolding and abstraction operators must be chosen to achieve the biggest optimization possible while ensuring termination and total correctness of the transformation. Moreover, in many cases we transform a rewrite theory whose operators obey structural, algebraic axioms such as associativity, commutativity, and unity, into a much simpler rewrite theory whose operators obey no axioms. This makes it possible to run such theories into an independent rewriting infrastructure that does not support rewriting modulo axioms. Furthermore, some costly analyses that may require significant (or even unaffordable) resources, both in time and space, can be now effectively performed after the transformation. This includes the analysis of cryptographic communication protocols that are currently handled by some ad-hoc combination of separate techniques and that can be recast as distinct instances of our generic partial evaluation scheme. See [1] for extra references on narrowing-driven partial evaluation.

After some brief preliminaries in Section 2, we sketch the specialization algorithm for rewrite theories in Section 3, which works in two phases: partial evaluation and compression refactoring. In Section 4, we provide a suitable unfolding operator dealing with theories that do not meet the finite variant property. Then, we discuss some preliminary experiments that show the optimization capability of our technique and we conclude.

2 Preliminaries

Let Σ be a *signature* that includes typed operators (also called function symbols) of the form $f : s_1 \dots s_m \to s$ where s_i, and s are sorts in a poset $(S, <)$ that models subsort relations (e.g. $s < s'$ means that sort s is a subsort of s'). Σ is assumed to be *preregular*, so each term t has a least sort under $<$, denoted $ls(t)$. Binary operators in Σ may have attached an axiom declaration that specifies any combinations of algebraic laws such as associativity (`assoc`), commutativity (`comm`), and identity (`id`). By $ax(f)$, we denote the set of algebraic axioms for the operator f. By $\mathcal{T}(\Sigma, \mathcal{X})$, we denote the usual non-ground term algebra built over Σ and the set of (typed) variables $\mathcal{X}$. By $\mathcal{T}(\Sigma)$, we denote the ground term algebra over Σ. By notation $x : s$ we denote a variable x with sort s. Any expression $\overline{t_n}$ denotes a finite sequence $t_1 \dots t_n$, $n \geq 0$, of terms. A *position* w in a term t is represented by a sequence of natural numbers that addresses a subterm of t (Λ denotes the empty sequence, i.e., the root position). Given a term t, we let $Pos(t)$ denote the set of positions of t. We denote the usual prefix preorder over positions by $\leq$. By $t_{|w}$, we denote the *subterm* of t at position w. By $root(t)$ we denote the operator of t at position Λ.

For an equational theory $\mathcal{E} = (\Sigma, E \uplus B)$ to be executable, its equations E must be *convergent* (i.e., confluent, sort-decreasing, terminating, and coherent modulo the given axioms B) [8]. This ensures: 1) that every input expression t has one (and only one) *canonical* form $t{\downarrow_{\mathcal{E}}}$; and 2) rewriting modulo the equations and axioms can be easily implemented by using the oriented equations of $\vec{E}$ as the only simplification rules, while the equations in B are just encapsulated within a powerful algorithm of pattern matching modulo B that is used at each rewrite step with $\vec{E}$. Throughout the paper, we assume that equational theories are always convergent. By $=_B$, we denote the equality relation given by the set of axioms B over $\mathcal{T}(\Sigma, \mathcal{X})$.

A *rewrite theory* is a triple $\mathcal{R} = (\Sigma, E \uplus B, R)$, where $(\Sigma, E \uplus B)$ is an equational theory and R is a set of rules of the form $l \Rightarrow r$, with $l, r \in \mathcal{T}(\Sigma, \mathcal{X})$, that may be non-confluent and non-terminating. Rewrite theories provide a natural computation model for concurrent systems as shown in the following example.

Example 1 *Let us consider a rewrite theory* $\mathcal{R} = (\Sigma, E \uplus B, R)$ *that encodes a close variant of the handshake network protocol of [14]. The theory models an environment where several clients and servers coexist. The signature* Σ *includes several operators and sorts that model protocol entities. Names of the sorts are self-explanatory: for example, servers are typed with sort* `Serv`*, clients with sort* `Cli`*, and messages with sort* `Message`*.*

Messages are encoded as non-empty, associative, sequences $\mathtt{s}_1 \ldots \mathtt{s}_n$ *where, for the sake of simplicity, each* $\mathtt{s}_i$ *is a term of sort* `Symbol` *in the alphabet* `{a,b,c}`*. We assume that* `Symbol < Message`*, hence any symbol is also a (one-symbol) message. Clients are represented as 5-tuples of the form* `[C,S,Q,K,V]` *of sort* `Cli`*, where* `C` *is the client's name,* `S` *is the name of the server* `C` *wants to communicate with,* `Q` *is a message encoding a client handshake request,* `K` *is a natural number (specified in Peano's notation) that determines an encryption/decryption key for messages, and* `V` *is a constant value that models the server handshake status. Initially, the status is set to the empty value* `mt`*, and it changes to* `success` *whenever the handshaking process succeeded. Servers are simply modeled by means of pairs of the form* `[S,K]` *of sort* `Serv`*, where* `S` *is a server name, and* `K` *is an encryption/decryption key. All network packets are represented as pairs of the form* `Host <- CNT` *of sort* `Packet`*, where* `Host` *is a client or server recipient, and* `CNT` *specifies the packet content. Specifically,* `CNT` *is a term* `{H,M}`*, with* `H` *being the sender's name and* `M` *being a message that represents either a client handshake request or a server response. System states are formalized as multisets* $\mathtt{t}_1 \,\&\, \ldots \,\&\, \mathtt{t}_m$ *of clients, servers, and network packets via the ACU operator* `_&_` *whose unity element is the constant* `null`*. The protocol dynamics is specified by the following three rules that implements a handshake protocol where clients and servers agree on a shared key* `K`*.*

```
rl [req] : [C,S,Q,K,mt] => (S <- {C,enc(Q,K)}) & [C,S,Q,K,mt] .
rl [reply] : (S <- {C,M}) & [S,K] => (C <- {S,dec(M,K)}) & [S,K] .
rl [rec] : (C <- {S,Q}) & [C,S,Q,K,mt] => [C,S,Q,K,success] .
```

More specifically, the rule `req` *allows a client* `C` *to start a handshake request with a sever* `S` *by sending an encrypted message* `enc(Q,K)` *to* `S` *so that the message* `Q` *is encrypted by using the client's key* `K`*. The rule* `reply` *lets the server* `S` *consume a client handshake request packet* `S <- {C,M}` *by first decrypting the incoming message* `M` *with the server key and then sending a response packet back to* `C` *that includes the decrypted request message. The rule* `rec` *successfully completes the handshake between* `C` *and* `S` *whenever the server response packet* `C <- {S,Q}` *includes a message* `Q` *which is equal to the initial client request message. In this case, the status of the client is changed from* `mt` *to* `success`*. Note that the handshake succeeds when the client and server use the same key* `K`*.*

Encryption and decryption capabilities are implemented by two functions (namely, `enc(M,K)` *and* `dec(M,K)`*) that are specified by the equational theory* $\mathcal{E}$ *in* $\mathcal{R}$*. The equational theory* $\mathcal{E}$ *implements a* Caesar *cipher with key* `K`*, which is a simple substitution cipher where each symbol in the plaintext message is replaced by the symbol* `K` *positions forward in the alphabet. The cipher is circular, i.e., it works modulo the cardinality of the alphabet. For instance,* `enc(a b,s(0))` *would deliver* `(b c)`*, and* `dec(a b,s(0))` *would be the message* `(c a)`*. The equational theory* $\mathcal{E}$ *includes the equations*[3] *in Figure 1. In the specification, the equational attribute* `variant` *is used to identify the equations to be considered by the folding variant narrowing strategy.*

[3] Due to the lack of space, we omitted the definition of the operators `[_,_,_]`, `_<_`, `_+_` that respectively implements the usual *if-then-else* construct, the *less-than* relation and the (associative and commutative) addition over natural numbers.

```
var M : Message .     var X K : Nat .     var S : Symbol .
eq toNat(a) = 0 [variant] .                 eq toSym(0)= a [variant] .
eq toNat(b) = toNat(a) + s(0) [variant] .   eq toSym(s(0)) = b [variant] .
eq toNat(c) = toNat(b) + s(0) [variant] .   eq toSym(s(s(0))) = c [variant] .
eq len = s(s(s(0))) --- Alphabet cardinality
eq shift(X) = [ s(X) < len,s(X), 0 ] [variant] .
eq unshift(0) = s(s(0)) [variant] .     eq unshift(s(X)) = X [variant] .
eq e(X,0) = X [variant] .               eq e(X,s(Y)) = e(shift(X),Y) [variant] .
eq d(X,0) = X [variant] .               eq d(X,s(Y)) = d(unshift(X),Y) [variant] .
eq enc(S,K) = toSym(e(toNat(S),K)) [variant] .
eq enc(S M,K) = toSym(e(toNat(S),K)) enc(M,K) [variant] .
eq dec(S,K) = toSym(d(toNat(S),K))[variant] .
eq dec(S M,K) = toSym(d(toNat(S),K)) dec(M,K) [variant] .
```

Figure 1: Equational theory $\mathcal{E}$ encoding the *Caesar* cipher.

3 Partial Evaluation of Rewrite Theories

In this section, we briefly present the specialization procedure NPER$^{\mathcal{U}}$ that allows a rewrite theory $\mathcal{R} = (\Sigma, E \uplus B, R)$ to be optimized by specializing the underlying equational theory $\mathcal{E} = (\Sigma, E \uplus B)$ with respect to the (calls in the) rewrite rules of R. The procedure NPER$^{\mathcal{U}}$ is parametric w.r.t. an unfolding operator $\mathcal{U}$ that is used to construct finite narrowing derivations for a given expression. NPER$^{\mathcal{U}}$ is based on a suitable extension of the equational, narrowing-driven partial evaluation algorithm for equational theories EqNPE$^{\mathcal{U}}$ of [1] shown below.

3.1 Partial Evaluation of Equational Theories

Given $\mathcal{E} = (\Sigma, E \uplus B)$ and a set Q of calls (henceforth called *specialized calls*), the main goal of EqNPE$^{\mathcal{U}}$ is to derive a new equational theory $\mathcal{E}'$ that computes the same answers (and values) for any input term that is a recursive instance (modulo axioms) of a term in Q. The procedure follows the style of Gallagher's partial deduction method [11], with two distinct control levels: the local level, which is controlled by an unfolding operator, and the global level, which is managed by an abstraction operator.

Unfolding. To partially evaluate $\mathcal{E}$ with respect to Q, the EqNPE$^{\mathcal{U}}$ algorithm starts by constructing in $\mathcal{E}$ a finite, possibly partial (folding variant) narrowing tree for each input term t of Q. This is done by using the unfolding operator $\mathcal{U}$ that determines when and how to stop the narrowing computations.

Abstraction. In order to guarantee that all possible executions for t in the original theory $\mathcal{E}$ are *covered* by the specialization, every (sub-)term in any leaf of the tree is required to be *equationally closed* w.r.t. Q. The equational closedness extends the classical PD closedness by recursing over the term structure (in order to handle nested function calls) and by considering B-equivalence of terms.

Roughly speaking, consider a natural partition[4] of the signature as $\Sigma = \mathcal{D} \uplus \mathcal{C}$, where the values computed by simplification (i.e., reduction to canonical form) with $\vec{E}$ modulo B are constructor terms, whereas the function symbols $f \in \mathcal{D}$ are viewed as defined functions that

[4]This distinction between constructor and defined symbols is more sophisticated than the standard division in the TRSs literature since Rewriting Logic supports overloaded symbols that can play both roles. Consider, e.g., the sort poset `Zero One < Nat` and the equation `s(s(X:Nat))=X:Nat`; in this setting, `s:Zero-> One` is a constructor symbol, whereas `s:Nat -> Nat` is a defined symbol.

are evaluated away by simplification with $\vec{E}$ modulo B. A term u is closed modulo B w.r.t. Q (we also say that u is Q-closed modulo B) iff either: (i) it does not contain defined function symbols of $\mathcal{D}$, or (ii) there exists a substitution θ and a (possibly renamed) $q \in Q$ such that $u =_B q\theta$, and the terms in θ are recursively Q-closed. For instance, given a defined binary symbol $\bullet$ that does not obey any structural axioms (in particular the commutativity), the term $t = a \bullet (Z \bullet a)$ is closed w.r.t. $Q = \{a \bullet X, Y \bullet a\}$ or $\{X \bullet Y\}$, but it is not with Q being $\{a \bullet X\}$; however, it would be closed if $\bullet$ were commutative.

Note that several iterations of i) and ii) may be needed because some of the leaves in deployed narrowing trees might include calls, i.e., (sub-)terms, that are not Q-closed modulo B. At each iteration, an abstraction operator is applied to properly add the uncovered (sub-)terms to the set of already partially evaluated calls, yielding a new set of terms which may need further evaluation. The process is iteratively repeated as far as new terms are introduced yet ensuring that the set cannot grow infinitely. The abstraction operator guarantees that only finitely many expressions are evaluated, thus ensuring global termination of the specialization.

Theory generation. The EqNPE$^{\mathcal{U}}$ algorithm does not explicitly compute a partially evaluated equational theory. It does so implicitly, by computing a (generally augmented) set Q' of partially evaluated terms that unambiguously determine the desired partially evaluated equations E as the set of *resultants* $t\sigma = t'$ associated with the derivations in the narrowing tree from the root $t \in Q'$ to the leaf t with computed answer substitution σ, such that the closedness condition modulo B w.r.t. Q' is satisfied for all function calls that appear in the right-hand sides of the equations in E'.

In the following, we assume the existence of the function GenTheory$(Q', (\Sigma, E \uplus B))$ that delivers the partially evaluated equational theory $\mathcal{E}' = (\Sigma', E' \uplus B')$ univocally determined by Q' and the original equational theory $\mathcal{E} = (\Sigma, E \uplus B)$.

3.2 The NPER$^{\mathcal{U}}$ scheme for the Specialization of Rewrite Theories

We first provide some auxiliary notions that are useful to describe the generic NPER$^{\mathcal{U}}$ scheme. Roughly speaking, the specialization of the rewrite theory $\mathcal{R} = (\Sigma, E \uplus B, R)$ is achieved by partially evaluating the hosted equational theory $\mathcal{E} = (\Sigma, E \uplus B)$ w.r.t. the rules of R, which is done by using the partial evaluation procedure EqNPE$^{\mathcal{U}}$ of Section 3.1. By providing suitable unfolding and abstraction operators, different instances of the specialization scheme can be defined. An unfolding operator that is able to deal with theories that do not meet the finite variant property is introduced in Section 4.

Given $\Sigma = \mathcal{D} \uplus \mathcal{C}$, let $\mathcal{D}_E$ be the set of the defined symbols of $\mathcal{D}$ that appear in the set of equations E. Given a term t, a *maximal function call* in t is a subterm $t_{|w}$ of t, with $w \in Pos(t)$, such that (i) $root(t_{|w}) \in \mathcal{D}_E$, and (ii) there does not exist $w' \in Pos(t)$, such that $w' \leq w$ and $t_{|w'} \in \mathcal{D}_E$. Given a rewrite rule $s \Rightarrow t$ of R, by $mcalls(s \Rightarrow t)$ we denote the set of all the maximal function calls that occur in s and t. Also, $mcalls(R)$ is the set of all maximal calls in the rules of R. The NPER$^{\mathcal{U}}$ procedure is outlined in Algorithm 1.

Roughly speaking, given the rewrite theory $\mathcal{R} = (\Sigma, E \uplus B, R)$, the procedure consists of two phases. Phase 1 applies the EqNPE$^{\mathcal{U}}$ algorithm to specialize the equational theory $\mathcal{E} = (\Sigma, E \uplus B)$ w.r.t. a set Q of specialized calls that consists of all of the maximal functions calls[5] that appear in the rewrite rules of R. This phase produces the new set of specialized calls Q' from which the partial evaluation $\mathcal{E}' = (\Sigma', E' \uplus B')$ of $\mathcal{E}$ w.r.t. Q is univocally derived by executing GenTheory$(Q', (\Sigma, E \uplus B))$.

[5] For simplicity, we assume that Q is normalized w.r.t. the equational theory $\mathcal{E}$. If this were not the case, we would add to the resulting specialization suitable "bridge" equations $t = t\downarrow_{\mathcal{E}}$, for each $t \in Q$ that is not in canonical form.

Algorithm 1 Symbolic Specialization of Rewrite Theories $\text{NPER}^{\mathcal{U}}(\mathcal{R})$

Require:
A rewrite theory $\mathcal{R} = (\Sigma, E \uplus B, R)$, an unfolding operator $\mathcal{U}$

1: **function** $\text{NPER}^{\mathcal{U}}(\mathcal{R})$
Phase 1. Partial Evaluation
2: $Q \leftarrow mcalls(R)$
3: $Q' \leftarrow \text{EqNPE}^{\mathcal{U}}((\Sigma, E \uplus B), Q)$
4: $\mathcal{E}' \leftarrow \textsc{GenTheory}(Q', (\Sigma, E \uplus B))$
Phase 2. Compression
5: $\mathcal{R}' \leftarrow \textsc{Compress}(\mathcal{R}, \mathcal{E}', Q')$
6: **return** $\mathcal{R}'$

Phase 2 is performed by the Compress post-processing, shown in Algorithm 2, that takes as input the original rewrite theory $\mathcal{R} = (\Sigma, E \uplus B, R)$, the computed partial evaluation $\mathcal{E}' = (\Sigma', E' \uplus B')$, and the final set of specialized calls Q' from which $\mathcal{E}'$ derives. The algorithm computes a new, much more compact equational theory $\mathcal{E}'' = (\Sigma'', E'' \uplus B'')$ where unused symbols and unnecessary repetition of variables are removed, and equations of E' are simplified by renaming similar expressions w.r.t. an independent renaming function ρ that is derived from set of specialized calls Q'.

Formally, for each t of sort s in Q' with $root(t) = f$, $\rho(t) = f_t(\overline{x_n : s_n})$, where $\overline{x_n}$ are the distinct variables in t in the order of their first occurrence and $f_t : \overline{s_n} \to s$ is a new function symbol that does not occur in Σ or Q' and is different from the root symbol of any other $\rho(t')$, with $t' \in Q'$ and $t' \neq t$. By abuse, we let $\rho(T)$ denote the set $T' = \{\rho(t) \mid t \in T\}$ for a given set of terms T.

Algorithm 2 Compression algorithm

Require:
A rewrite theory $\mathcal{R} = (\Sigma, E \uplus B, R)$, a partial evaluation $\mathcal{E}' = (\Sigma', E' \uplus B')$ of $(\Sigma, E \uplus B)$ w.r.t. a set of specialized calls Q.

1: **function** $\textsc{Compress}(\mathcal{R}, \mathcal{E}', Q)$
2: Let ρ be an independent renaming for Q in
3: $E'' \leftarrow \bigcup_{t \in Q} \{\rho(t)\theta = RN_\rho(t') \mid t\theta = t' \in E'\}$
4: $R' \leftarrow \{RN_\rho(l) \Rightarrow RN_\rho(r) \mid l \Rightarrow r \in R\}$
5: $\Sigma'' \leftarrow (\Sigma' \setminus \{f \mid f \text{ occurs in } ((E \uplus B) \setminus (E' \uplus B'))\}) \cup \{root(\rho(t)) \mid t \in Q\}$
6: $B'' = \{ax(f) \in B' \mid f \in \Sigma' \cap \Sigma''\}$
7: **return** $(\Sigma'', E'' \uplus B'', R')$

where

$$RN_\rho(t) = \begin{cases} c(\overline{RN_\rho(t_n)}) & \text{if } t = c(\overline{t_n}) \text{ with } c : \overline{s_n} \to s \in \Sigma \text{ s.t. } c \in \mathcal{C},\ ls(t) = s,\ n \geq 0 \\ \rho(u)\theta' & \text{if } \exists \theta, \exists u \in Q \text{ s.t. } t =_B u\theta \text{ and } \theta' = \{x \mapsto RN_\rho(x\theta) \mid x \in Dom(\theta)\} \\ t & \text{otherwise} \end{cases}$$

Essentially, the Compress algorithm of Figure 2 can be seen as a refactoring transformation that recursively computes, by means of the function RN_ρ, a new equation set E'' by replacing each call in E' by a call to the corresponding renamed function according to ρ. Furthermore, a new rewrite rule set R' is also produced by consistently applying RN_ρ to the rewrite rules of R. Formally, each rewrite rule $l \Rightarrow r$ in R is transformed into the rewrite rule $RN_\rho(l) \Rightarrow RN_\rho(r)$, in which every maximal function call t in the rewrite rule is recursively renamed according to

the independent renaming ρ. The algorithm also computes the specialized signature Σ'' and restricts the set B' to those axioms obeyed by the function symbols in $\Sigma' \cap \Sigma''$. Finally, the rewrite theory $\mathcal{R}' = (\Sigma'', E'' \uplus B'', R')$ is delivered as the final outcome.

Note that, while the independent renaming suffices to rename the left-hand sides of the equations in E' (since they are mere instances of the specialized calls), the right-hand sides are renamed by means of the auxiliary function RN_ρ, which recursively replaces each call in the given expression by a call to the corresponding renamed function (according to ρ).

4 Instantiating the Specialization Scheme for Rewrite Theories

In this section, we formulate an instance of the generic specialization scheme of Section 3 by providing a concrete implementation $\mathcal{U}_{\trianglelefteq}$ of the generic unfolding operator $\mathcal{U}$ that implements the local control and is based on folding variant narrowing. Since termination of folding variant narrowing is not generally guaranteed, the unfolding operator $\mathcal{U}_{\trianglelefteq}$ must incorporate some mechanism to stop the construction of the narrowing trees. For this purpose, a number of standard techniques can be applied, including depth-bounds, loop-checks, well-founded orderings, well-quasi orderings, etc. Within the narrowing-driven approach, unfolding rules have been traditionally defined by using a particular type of well-quasi ordering: homeomorphic embedding. This is why we formulate $\mathcal{U}_{\trianglelefteq}$ by relying on an equational, order-sorted extension $\trianglelefteq$ of the classical homeomorphic embedding relation defined in [2] that detects the risk of non-termination when a term is reached that embeds a previous term of the same derivation.

Roughly speaking, a homeomorphic embedding relation is a structural preorder under which a term t is greater than (i.e., it embeds) another term t', written as $t' \trianglelefteq t$, if t' can be obtained by deleting some parts of t, e.g., $s(s(X + Y) * (s(X) + Y))$ embeds $s(Y * (X + Y)))$. When iteratively computing a sequence $t_1, t_2, \ldots, t_n$, finiteness of the sequence can be guaranteed by using the embedding as a whistle: whenever a new expression t_{n+1} is to be added to the sequence, we first check whether t_{n+1} embeds any of the expressions already in the sequence. If that is the case, we say that $\trianglelefteq$ whistles, i.e., it has detected (potential) non-termination and the computation has to be stopped. Otherwise, t_{n+1} can be safely added to the sequence and the computation can proceed.

As for the global level of control, it is enforced by means of an abstraction operator that is based on an equational order sorted extension of the least general generalization algorithm of [3], and it guarantees that the number of unfolded narrowing trees is kept finite. Computing a *least general generalization* (lgg) for two expressions t_1 and t_2, also known as *least general anti-unifier*, means finding the least general expression t such that both t_1 and t_2 are instances of t under appropriate substitutions. Due to the algebraic axioms, in general there can be more than one least general generalizer of two expressions. As a simple example, we record the travel history of a person using a list (with associative list constructor symbol '.') that is ordered by the chronology in which the visits were made; e.g., `paris.paris.bonn.nyc` denotes that `paris` has been visited twice before visiting `bonn` and then `nyc`. The travel histories `paris.paris.bonn.nyc` and `bonn.bonn.rome` have two incomparable least general generalizers modulo axioms (a) `L1.bonn.L2` and (b) `C.C.L`, meaning that (a) the two travelers visited `bonn`, and (b) they consecutively repeated a visit to their own first city. Note that the two generalizers are least general and incomparable, since neither of them is an instance of the other modulo axioms.

Example 2 *Consider a specific instance of the rewrite theory of Example 1 where servers and clients reach consensus on a pre-shared fixed key; for simplicity assume* `K=s(s(0))`. *Let* $\mathcal{R} = (\Sigma, E \uplus B, R)$ *be such a rewrite theory, where* $\mathcal{E} = (\Sigma, E \uplus B)$ *is the equational theory of* $\mathcal{R}$. *In* $\mathcal{E}$, *the FVN trees*

associated to encryption and decryption capabilities may be infinite. For instance, the FVN tree for the call enc(M,s(s(0))) *is infinite since the message* M *may have an arbitrary size. In fact, terms of the form* ($t_1 \ldots t_n$ enc(M′,s(s(0)))) *can be narrowed from* enc(M,s(s(0))), *where* enc(M′,s(s(0))) *can be further narrowed to unravel an unlimited sequence of identical terms modulo renaming. Nonetheless homeomorphic embedding detects this non-terminating behaviour since* enc(M′,s(s(0))) *embeds* enc(M,s(s(0))).

By using the unfolding operator $\mathcal{U}_{\trianglelefteq}$, *the first phase of the* NPER$^{\mathcal{U}}(\mathcal{R})$ *Algorithm 1 computes the initial set* Q = {enc(M,s(s(0))), dec(M,s(s(0))} *consisting of the maximal functional calls in* $\mathcal{R}$. *Then, the equational theory* $\mathcal{E}$ *is partially evaluated by* EqNPE$^{\mathcal{U}_{\trianglelefteq}}$ *w.r.t.* Q. *During the partial evaluation process,* $\mathcal{U}_{\trianglelefteq}$ *only unravels finite fragments of the FVN narrowing trees that are rooted by the specialized calls, thereby yielding the partial evaluation* $\mathcal{E}'$ *of* $\mathcal{E}$ *in Figure 2.*

```
eq dec(a,s(s(0))) = b [variant] .     eq dec(b,s(s(0))) = c [variant] .
eq dec(c,s(s(0))) = a [variant] .
eq dec(S:Symbol M:Message, s(s(0))) = toSym(unshift(unshift(toNat(S:Symbol))))
                                                    dec(M:Message, s(s(0))) [variant] .
eq enc(a,s(s(0))) = c [variant] .     eq enc(b,s(s(0))) = a [variant] .
eq enc(c,s(s(0))) = b [variant] .
eq enc(S:Symbol M:Message, s(s(0))) =
    toSym([[toNat(S:Symbol) < s(s(0)),s(toNat(S:Symbol)),0] < s(s(0)),
    s([toNat(S:Symbol) < s(s(0)),s(toNat(S:Symbol)),0]),0]) enc(M:Message, s(s(0))) [variant] .
eq toSym([[toNat(a) < s(s(0)),s(toNat(a)),0] <
                              s(s(0)),s([toNat(a) < s(s(0)),s(toNat(a)),0]),0]) = c [variant] .
eq toSym([[toNat(b) < s(s(0)),s(toNat(b)),0] <
                              s(s(0)),s([toNat(b) < s(s(0)),s(toNat(b)),0]),0]) = a [variant] .
eq toSym([[toNat(c) < s(s(0)),s(toNat(c)),0] <
                              s(s(0)),s([toNat(c) < s(s(0)),s(toNat(c)),0]),0]) = b [variant] .
eq toSym(unshift(unshift(toNat(a)))) = b [variant] .
eq toSym(unshift(unshift(toNat(b)))) = c [variant] .
eq toSym(unshift(unshift(toNat(c)))) = a [variant] .
```

Figure 2: NPER$^{\mathcal{U}_{\trianglelefteq}}$ Phase 1: Partial evaluation of $\mathcal{E}$ w.r.t. Q

The second phase of the algorithm produces the compressed equational theory $\mathcal{E}''$ *of Figure 3 by computing the following renaming for the theory functions. This greatly simplifies* $\mathcal{E}'$ *since it gets rid of long sequences of nested calls and non-variable function arguments.*

dec(M : Message, s(s(0))) ↦ f0(M : Message)
enc(M : Message, s(s(0))) ↦ f1(M : Message)
toSym(unshift(unshift(toNat(X : Symbol)))) ↦ f3(X : Symbol)
toSym([[toNat(X : Symbol) < s(s(0)), s(toNat(X : Symbol)), 0] < s(s(0)),
s([toNat(X : Symbol) < s(s(0)), s(toNat(X : Symbol)), 0]), 0]) ↦ f2(X : Symbol)

Finally, it is worth noting that the resulting specialization $\mathcal{E}''$ *provides a highly optimized version of* $\mathcal{E}$ *for an arbitrarily fixed key* K=s(s(0)), *where both functional and structural compression are achieved. Specifically, data structures in* $\mathcal{E}$ *for natural numbers and their associated operations for message encryption and decryption are totally removed from* $\mathcal{E}''$. *Note that the* _+_, *operator together with its associative and commutative axioms, disappears from* $\mathcal{E}''$, *thereby avoiding expensive matching operations modulo axioms. Encryption in* $\mathcal{E}''$ *(resp., decryption) is now a direct mapping* f0 *(resp.,* f1*) that associates messages to their corresponding crypted (resp. decrypted) counterparts, avoiding a huge amount of computation in the profuse domain of natural numbers. Finally, the computed renaming is also applied to* $\mathcal{R}$ *by respectively replacing the maximal function calls* enc(M,s(s(0))) *and* dec(M,s(s(0))) *with* f0(M) *and* f1(M) *into the rewrite rules of* $\mathcal{R}$. *This al-*

```
eq f0(a) = b [variant] .    eq f0(b) = c [variant] .    eq f0(c) = a [variant] .
eq f2(a) = c [variant] .    eq f2(b) = a [variant] .    eq f2(c) = b [variant] .
eq f1(a) = c [variant] .    eq f1(b) = a [variant] .    eq f1(c) = b [variant] .
eq f3(a) = b [variant] .    eq f3(b) = c [variant] .    eq f3(c) = a [variant] .
eq f0(S:Symbol M:Message) = f3(S:Symbol) f0(M:Message) [variant] .
eq f1(S:Symbol M:Message) = f2(S:Symbol) f1(M:Message) [variant] .
```

Figure 3: NPER$^{\mathcal{U}_{\trianglelefteq}}$ Phase 2: Compression of $\mathcal{E}'$

lows the (renamed) rewrite rules to be able to access the new specialized encryption and decryption functionality provided by $\mathcal{E}''$.

Note that our methodology may, in some cases, transform a rewrite theory whose equational theory does not satisfy the FVP, into one that does. This allows narrowing-based reachability problems to be solved in the specialized program, whereas it is not possible into the original one. For instance, by restricting the handshake protocol to messages of a fixed size (e.g. 3 symbols), we could get a specialization that meets the FVP, in which the following reachability goal `[Cli-A,Srv-A,Q,K,mt] & [Srv-A,K] & (Srv-A <- {Cli-A,abc}) =>* [Srv-A,K] & [Cli-A,Srv-A,Q,K,success]` can be solved. The solution allows to infer the client key `K=s(s(0))` and the non-encrypted message `Q=(bca)` in the initial state whenever the crypted message abc is sent to the server.

The NPER$^{\mathcal{U}_{\trianglelefteq}}$ specialization algorithm has been implemented in a prototype system that is available at [16]. Table 1 contains the experiments that we have performed using an Intel Xeon E5-1660 3.3GHz CPU with 64 GB RAM running Maude v3.0 and considering the average of ten executions for each test. These experiments together with the source code of all examples are also publicly available at [16]. We have considered two variants of the handshake protocol previously discussed in the paper for input messages of three different sizes: one hundred thousand symbols, five hundred thousand symbols, and one million symbols (Column M_{size}). The two variants differ in the introduction of an extra function (Fibonacci) in the underlying equational theory to make the key generation heavier, and in this case, we introduce a generous time bound to stop the execution after a substantial number of rewrites. We have benchmarked the original rewrite theory $\mathcal{R}$ and the specialized rewrite theory $\mathcal{R}'$ on these data. We do not explicitly show the specialization times since they are negligible for all problems (< 100 ms). For each benchmark, the number of rewrites for a common initial state in each rewrite theory is shown in columns $\#Rews_{\mathcal{R}}$ and $\#Rews_{\mathcal{R}'}$, respectively. The percentage of reduction in terms of number of rewrites is shown in the *Reduction* column.

The relative speedups that we achieved thanks to specialization are given in the *Speedup* column and computed as the ratio $T_{\mathcal{R}}/T_{\mathcal{R}'}$. Our figures show that the specialized theories achieve a significant improvement in execution time when compared to the original rewrite theory, with an average speedup for these benchmarks of 3.47.

We are currently working on improving our prototype and on more powerful instances of our scheme for relevant classes of rewrite theories, such as those that satisfy FVP and theories that protect a constructor sub-theory but are non-FVP so that they cannot be automatically optimized by existing techniques.

References

[1] M. Alpuente, A. Cuenca-Ortega, S. Escobar & J. Meseguer (2020): *A Partial Evaluation Framework for Order-Sorted Equational Programs modulo Axioms*. *JLAMP* 110.

	M_{size}	$\#Rews_{\mathcal{R}}$	$\#Rews_{\mathcal{R}'}$	*Reduction*	$T_{\mathcal{R}}$ (ms)	$T_{\mathcal{R}'}$ (ms)	*Speedup*
Handshake Protocol	100K	2,600,115	400,002	84.62%	221	96	2.30
w/o Fibonacci	500K	13,000,205	2,000,002	84.62%	1,950	731	2.67
(success)	1M	26,000,100	4,000,002	84.62%	5,137	2,191	2.34
Handshake Protocol	100K	92,003,651	10,000,051	89.13%	10,200	1,716	5.94
with Fibonacci	500K	442,003,651	50,000,051	88.69%	53,424	12,185	4.38
(time bound)	1M	879,503,651	100,000,051	88.63%	129,112	40,857	3.16

Table 1: Experimental results for the specialization of the Handshake protocol

[2] M. Alpuente, A. Cuenca-Ortega, S. Escobar & J. Meseguer (2020): *Order-sorted Homeomorphic Embedding modulo Combinations of Associativity and/or Commutativity Axioms*. Fundamenta Informaticae. To appear.

[3] M. Alpuente, S. Escobar, J. Espert & J. Meseguer (2014): *A Modular Order-Sorted Equational Generalization Algorithm*. Information and Computation 235, pp. 98–136.

[4] M. Alpuente, M. Falaschi & G. Vidal (1998): *Partial Evaluation of Functional Logic Programs*. ACM Transactions on Programming Languages and Systems 20(4), pp. 768–844.

[5] R. M. Burstall & J. Darlington (1977): *A Transformation System for Developing Recursive Programs*. Journal of the ACM 24(1), pp. 44–67.

[6] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer & C. Talcott (2007): *All About Maude: A High-Performance Logical Framework*.

[7] H. Comon-Lundh & S. Delaune (2005): *The Finite Variant Property: How to Get Rid of Some Algebraic Properties*. In: Proc. of RTA 2005, LNCS 3467, Springer, pp. 294–307.

[8] F. Durán & J. Meseguer (2012): *On the Church-Rosser and Coherence Properties of Conditional Order-sorted Rewrite Theories*. JLAP 81(7–8), pp. 816–850.

[9] S. Escobar & J. Meseguer (2007): *Symbolic Model Checking of Infinite-State Systems Using Narrowing*. In: Proc. of RTA 2007, LNCS 4533, Springer, pp. 153–168.

[10] S. Escobar, R. Sasse & J. Meseguer (2012): *Folding Variant Narrowing and Optimal Variant Termination*. JLAP 81(7–8), pp. 898–928.

[11] J. P. Gallagher (1993): *Tutorial on Specialisation of Logic Programs*. In: Proc. of PEPM 1993, ACM, pp. 88–98.

[12] N. D. Jones, C. K. Gomard & P. Sestoft (1993): *Partial Evaluation and Automatic Program Generation*. Prentice-Hall.

[13] J. W. Lloyd & J. C. Shepherdson (1991): *Partial Evaluation in Logic Programming*. The Journal of Logic Programming 11(3-4), pp. 217–242.

[14] J. Meseguer (2008): *The Temporal Logic of Rewriting: A Gentle Introduction*. In: Concurrency, Graphs and Models, LNCS 5065, Springer, pp. 354–382.

[15] J. Meseguer & P. Thati (2007): *Symbolic Reachability Analysis Using Narrowing and its Application to Verification of Cryptographic Protocols*. Higher-Order and Symbolic Computation 20(1–2), pp. 123–160.

[16] (2020): *The* Presto *Website*. Available at: `http://safe-tools.dsic.upv.es/presto`.

[17] J. R Slagle (1974): *Automated Theorem-Proving for Theories with Simplifiers, Commutativity, and Associativity*. Journal of the ACM 21(4), pp. 622–642.