Optimization of Rewrite Theories by Equational Partial Evaluation [⋆]

M. Alpuente[1], D. Ballis[2], S. Escobar[1], and J. Sapiña[1]

[1] VRAIN, Universitat Politècnica de València, Valencia, Spain
{alpuente,sescobar,sapina}@upv.es
[2] DMIF, Università degli Studi di Udine, Udine, Italy
demis.ballis@uniud.it

Abstract. In this paper, we develop an automated optimization framework for rewrite theories that supports sorts, subsort overloading, equations and algebraic axioms with free/non-free constructors, and rewrite rules modeling concurrent system transitions whose state structure is defined by means of the equations. The main idea of the framework is to make the system computations more efficient by partially evaluating the equations to the specific calls that are required by the transition rules. This can be particularly useful for automatically optimizing rewrite theories that contain overly general equational theories which perform unnecessary and costly computations involving pattern matching and/or unification modulo equations and axioms. The transformation is based on a suitable unfolding operator parameter that relies on the symbolic operational engine of Maude's equational theories, called *folding variant narrowing*, together with a generic abstraction operator. Depending on the properties of the rewrite theory, the unfolding and abstraction operators must be fine-tuned to achieve the biggest optimization possible while ensuring termination and total correctness of the transformation. We formalize two instances of our scheme for the case when the rewrite theory either has an infinite number of most general variants or a finite number of most general variants. Finally, we discuss some experimental results which demonstrate that the proposed optimization technique pays off in practice.

1 Introduction

Rewriting Logic (RWL) is a logic of change that extends 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 [23]. Roughly speaking, a rewrite theory $\mathscr{R} = (\Sigma, E \uplus B, R)$ seamlessly combines a *term rewriting system* (TRS) R, which specifies the system dynamics, with an *equational theory* $\mathscr{E}$ that defines the static structure of the system states as terms of an algebraic datatype. Given a signature Σ of program operators together with their type definition, the equational theory $\mathscr{E} = (\Sigma, E \uplus B)$ combines, in turn, a set E of equations (that are implicitly oriented from left to right and operationally used as simplification rules) on Σ and a set B of commonly occurring axioms such as associativity, commutativity, and identity that are essentially used for B-matching[3] (and are implicitly expressed in Maude as operator attributes).

[⋆] This research was partially supported by TAILOR, a project funded by EU Horizon 2020 research and innovation programme under GA No 952215, 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.

[3] For example, assuming a commutative binary operator $*$, the term $s(0) * 0$ matches within the term $X * s(Y)$ *modulo* the commutativity of symbol $*$ with matching substitution $\{X/0, Y/0\}$.

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 [34]. PE has been widely applied to a variety of programming paradigms, including functional programming [34] and logic programming [36], where it is usually called *partial deduction* (PD). In contrast to classical PE, partial deduction allows to not only instantiate input variables with constant values but also with terms that may contain variables, thus providing extra capabilities for specialization [36, 37].

Narrowing is a *symbolic* mechanism that extends term rewriting by replacing pattern matching with unification [32, 51]. The *Equational Narrowing-driven Partial Evaluation* (EQNPE) scheme of [6] extends PD to the specialization of order-sorted equational theories with respect to a set of input terms by making use of *folding variant narrowing* (also called *FV-narrowing* [31]). Thanks to the combined (logic and functional) capabilities of narrowing, the achieved transformation is strictly more powerful than the PE of both logic programs and functional programs [17]. In the EQNPE scheme, the key ingredients of PD get generalized to an order-sorted typed setting modulo axioms by formalizing: 1) a narrowing-based *unfolding operator* that ensures correction of the transformation; 2) *order-sorted equational homeomorphic embedding* for local termination (i.e., finiteness of unfolding); 3) *order-sorted equational closedness* (a recursive notion ensuring 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 anti-unification*) for global termination of the whole specialization process.

While the EqNPE scheme of [6] only applies to deterministic and terminating equational theories, partial evaluation has never been investigated in the context of non-deterministic and non-terminating *rewrite theories*. This paper addresses the specialization of such rewrite theories $\mathscr{R} = (\Sigma, E \uplus B, R)$, whose rewrite rules R are defined on top of an underlying equational theory $\mathscr{E} = (\Sigma, E \uplus B)$. Altogether, the rewrite theory $\mathscr{R}$ models a concurrent system that evolves by rewriting the system states by means of *equational rewriting*, i.e., rewriting with the rewrite rules of R *modulo* the equations and axioms of $\mathscr{E}$ [38]. To be executable in Maude, the equational theory $\mathscr{E}$ is required to be *convergent* (i.e., the equations of E are confluent, terminating, and sort-decreasing) and coherent modulo B. This ensures that every input expression t has one (and only one) *canonical* form $t{\downarrow_{\vec{E},B}}$ up to B-equality. On the other hand, the rules of R are required to be coherent w.r.t. $\mathscr{E}$, which allows the rewrite steps with $\mathscr{R}$ to always be postponed in favor of deterministically rewriting with $\mathscr{E}$.

In Maude, rewrite theories can also be *symbolically* executed by narrowing at *two levels*: (i) narrowing with the (typically non-confluent and non-terminating) rules of R modulo $\mathscr{E} = (\Sigma, E \uplus B)$; and (ii) narrowing with the (explicitly) oriented equations $\vec{E}$ modulo the axioms B. They both have practical applications: (i) narrowing with R modulo $\mathscr{E} = (\Sigma, E \uplus B)$ is useful for solving *reachability goals* [43] and *logical model checking* [29]; and (ii) narrowing with $\vec{E}$ modulo B is useful for $\mathscr{E}$-unification and variant computation[4] [31]. Both levels of narrowing should meet some conditions: (i) narrowing with R modulo $\mathscr{E}$ is performed in a "topmost" way (i.e., the rules in R rewrite the global system state) and there must be a finitary equational unification algorithm for $\mathscr{E}$; and (ii) narrowing with $\vec{E}$ modulo B requires that B is a theory with a finitary unification algorithm and that $\mathscr{E}$ is convergent. When $(\Sigma, E \uplus B)$ additionally has the

[4] A *variant* [24] of a term t in the theory $\mathscr{E}$ is the irreducible form of $t\sigma$ in $\mathscr{E}$ for a given substitution σ; in symbols, it is represented as the pair $(t\sigma{\downarrow_{\vec{E},B}}, \sigma)$.

property that a finite complete set of most general *variants* exists for each term, known as the *finite variant property* (FVP), $\mathscr{E}$-unification is finitary and *topmost* narrowing with R modulo the equations and axioms can be effectively performed. For variant computation and (variant-based) $\mathscr{E}$-unification, the folding variant narrowing[5] (or FV-narrowing) strategy of [31] is used in Maude, whose termination is guaranteed for theories that satisfy the FVP (also known as *finite variant theories*). Many relevant theories have the FVP, including theories of interest for Boolean satisfiability and theories that give algebraic axiomatizations of cryptographic functions used in communication protocols.

Partial evaluation techniques typically remove some computation states by performing as much program computation as possible, hence contracting the search space because some transitions are removed. However, narrowing-based analysis of rewrite theories generally requires the whole search space of a rewrite theory to be analyzed (i.e., all system states and transitions). Given the rewrite theory $\mathscr{R} = (\Sigma, E \uplus B, R)$, to avoid hindering said analysis, our method proceeds by specializing the underlying order-sorted equational theory $\mathscr{E} = (\Sigma, E \uplus B)$ to the precise use that the rules of R make of the functions that are defined in $\mathscr{E}$. This is done by partially evaluating $\mathscr{E}$ with respect to the *maximal* (or outermost) function calls that can be retrieved from the rules of R, in such a way that $\mathscr{E}$ gets rid of any possible overgenerality. Actually, while the transformation highly contracts the system states and, more generally, the functional computations given by $\mathscr{E}$ are greatly compacted, no system state disappears. 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 now be effectively performed after the transformation.

A preliminary version of this work was presented in [5].

Our Contribution The main contributions of this paper are as follows.

1. We formalize a specialization scheme for rewrite theories that extends the equational, narrowing-driven partial evaluation approach of [6] to the specialization of rewrite theories.

2. We ascertain the key requirements to be satisfied by the rewrite theory to guarantee the formal properties of our framework and we provide full proofs of all technical results in the article, which include preservation of executability conditions, termination, and strong correctness.

3. The original EqNPE framework of [6] was originally designed to deal with free constructors for which no equality relation can be established between any two different constructor symbols. This limitation was due to the fact that we did not consider *subsort overloading*

[5] The main idea of folding variant narrowing is to "fold" the search space of all FV-narrowing computations by using subsumption modulo B. That is, folding variant narrowing avoids computing any variant that is a substitution instance modulo B of a more general variant. Note that this notion is quite different from the classical folding operation of Burstall and Darlington's fold/unfold transformation scheme [21, 16], 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.

(i.e., the overloading of operators that are related in the subsort ordering) for constructor symbols. To deal with it, in this article we naturally extend some key notions of the EqNPE framework, including the definitions of equational closedness and equational abstraction. This allows us to deal with non-free constructors as overloaded function symbols that may behave as a constructor operator for some typing, while behaving as a defined function symbol for a higher typing. In line with the extended definitions, we have correspondingly extended our technical results in [6] for dealing with sorts and subsorts in a finer way.

4. Similarly to [6], for the equational component of the rewrite theory, our specialization algorithm follows the classic control strategy of logic specializers [37], with two separate control levels: 1) local control (managed by a generic unfolding operator) that avoids infinite evaluations and is responsible for the construction of the residual equations for each specialized call; and 2) global control or control of polyvariance (managed by a generic abstraction operator) that avoids infinite iterations of the partial evaluation algorithm and decides which specialized functions appear in the transformed theory. To further optimize both rules and equations, we introduce a final, post-processing compression transformation that highly contracts the system states and the functional computations occurring in the specialized rewrite theory.

5. We provide two different implementations of the unfolding operator based on FV-narrowing that adapt the generic technique to the FVP behavior of the equational theory $\mathcal{E}$ by distinguishing two cases:

   (a) $\mathcal{E}$ fulfills the finite variant property: since FV-narrowing trees are always finite in finite variant theories for any input term, the unfolding strategy is formulated as a process of total evaluation where the defined functions that have the FVP which appear in right-hand sides of rules are completely evaluated by computing a complete set of most general variants.

   (b) $\mathcal{E}$ does not satisfy the finite variant property: in this case, a subsumption check is performed at each FV-narrowing step that compares (under order-sorted equational homeomorphic embedding [6]) the current term with all previous narrowing redexes in the same derivation so that all infinite FV-narrowing computations are safely stopped.

6. We have implemented an experimental prototype system called Presto, and we provide an empirical evaluation of the system on a set of benchmark problems that test the speedups achieved for both rewriting and narrowing computations.

Plan of the paper The paper is organized as follows. In Section 2, we recall some preliminary notions and we provide the specification of a client-server communication protocol which is used as running example throughout the paper. The generic specialization scheme for rewrite theories is described in Section 3. After introducing the folding variant narrowing strategy in Section 4, in Section 5 we instantiate the specialization scheme for the two classes of equational theories already mentioned: theories that fulfill the FVP and theories that do not fulfill the FVP. The proposed scheme instantiations come with some non-trivial examples that highlight the power of our specialization methodology. Section 6 provides an experimental evaluation in the Presto system, which implements the proposed specialization framework. Our benchmarks demonstrate the program optimization that is achieved for narrowing as well as for rewriting computations. In Section 7, we discuss some related work and we conclude. Proofs of the main results are given in Appendix A, while Appendix B provides the full specification of the client-server communication protocol.

2 Preliminaries

Let Σ be a *signature* that includes typed operators (also called function symbols) of the form $f\colon s_1 \ldots s_m \to s$, where s_i, for $i = 1, \ldots n$, 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 unique least sort under $<$, denoted $ls(t)$. The connected components of $(S, <)$ are the equivalence classes $[s]$ corresponding to the least equivalence relation $\equiv_<$ containing $<$. For technical reasons, it is useful to assume that Σ has no ad-hoc overloading.[6] However, this assumption entails no real loss of generality: any Σ can be transformed into a semantically equivalent signature with no ad-hoc overloading (by symbol renaming). Note that avoiding ad-hoc overloading ensures that Σ is *sensible*, in the sense that for any two typings $f : s_1 \ldots s_n \to s$ and $f : s'_1 \ldots s'_n \to s'$ of a n-ary function symbol f, if s_i and s'_i are in the same connected component of $(S, <)$ for $1 \leq i \leq n$, then s and s' are also in the same connected component; this provides the right notion of *unambiguous* signature at the order-sorted level. 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 of terms $t_1 \ldots t_n$, $n \geq 0$. 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.

A *substitution* σ is a sorted mapping from a finite subset of $\mathcal{X}$ to $\mathcal{T}_\Sigma(\mathcal{X})$. Substitutions are written as $\sigma = \{X_1 \mapsto t_1, \ldots, X_n \mapsto t_n\}$. The identity substitution is denoted by *id*. Substitutions are homomorphically extended to $\mathcal{T}_\Sigma(\mathcal{X})$. The application of a substitution σ to a term t is denoted by $t\sigma$. The restriction of σ to a set of variables $V \subset \mathcal{X}$ is denoted $\sigma_{|V}$. Composition of two substitutions is denoted by $\sigma\sigma'$ so that $t(\sigma\sigma') = (t\sigma)\sigma'$.

A Σ-*equation* (or simply equation, where Σ is clear from the context) is an unoriented pair $\lambda = \rho$, where $\lambda, \rho \in \mathcal{T}_{\Sigma,\mathsf{s}}(\mathcal{X})$ for some sort $\mathsf{s} \in \mathsf{S}$, where $\mathcal{T}_{\Sigma,\mathsf{s}}(\mathcal{X})$ is the set of terms of sort s built over Σ and $\mathcal{X}$. An equational theory $\mathcal{E}$ is a pair $(\Sigma, E \uplus B)$ that consists of a signature Σ, a set E of Σ-equations, and a set B of equational axioms (e.g., associativity, commutativity, and/or identity) declared for some binary operators in Σ. The equational theory $\mathcal{E}$ induces a congruence relation $=_\mathcal{E}$ on $\mathcal{T}_\Sigma(\mathcal{X})$.

A term t is more general than (or at least as general as) t' modulo $\mathcal{E}$, denoted by $t \leq_\mathcal{E} t'$, if there is a substitution γ such that $t' =_\mathcal{E} t\gamma$. We also define $t \simeq_\mathcal{E} t'$ iff $t \leq_\mathcal{E} t'$ and $t' \leq_\mathcal{E} t$. By abuse of notation, we write $\leq_B$ and $\simeq_B$ when B is an axiom set.

[6] Given the overloaded operator $f : s_1 \ldots s_m \to s_0$ and $f : s'_1 \ldots s'_n \to s'_0$, subsort overloading means that $m = n$ and, for all i, $0 \leq i \leq n$, s_i and s'_i belong to the same connected component. Otherwise, the overloading of f is called ad-hoc.

A substitution θ is more general than (or at least as general as) σ modulo $\mathscr{E}$, denoted by $\theta \leq_{\mathscr{E}} \sigma$, if there is a substitution γ such that $\sigma =_{\mathscr{E}} \theta\gamma$, i.e., for all $x \in \mathscr{X}, x\sigma =_{\mathscr{E}} x\theta\gamma$. Also, $\theta \leq_{\mathscr{E}} \sigma\ [V]$ iff there is a substitution γ such that, for all $x \in V$, $x\sigma =_{\mathscr{E}} x\theta\gamma$.

An $\mathscr{E}$*-unifier* for a Σ-equation $t = t'$ is a substitution σ such that $t\sigma =_{\mathscr{E}} t'\sigma$. By $CSU_{\mathscr{E}}(t = t')$, we denote a *complete* set of $\mathscr{E}$-unifiers for the equation $t = t'$ so that any $\mathscr{E}$-unifier of $t = t'$ is less general modulo $\mathscr{E}$ than (at least) one element in the set.

A *rewrite rule* (or simply rule) is an expression of the form $\lambda \Rightarrow \rho$, where $\lambda, \rho \in \mathscr{T}_{\Sigma}(\mathscr{X})$. A rule $\lambda \Rightarrow \rho$ is *sort-decreasing* if $ls(\rho) \equiv_{<} ls(\lambda)$. A *rewrite theory* is a triple $\mathscr{R} = (\Sigma, E \uplus B, R)$, where $(\Sigma, E \uplus B)$ is an equational theory and R is a set of rewrite rules. A rewrite theory $(\Sigma, E \uplus B, R)$ is called *topmost* if there is a sort *State* such that: (i) for each rule $\lambda \Rightarrow \rho$, $ls(\lambda) \equiv_{<} State$ and $ls(\rho) \equiv_{<} State$; and (ii) there is no symbol $f \colon t_1 \dots t_n \to s \in \Sigma$ and $i \in \{1, \dots, n\}$ such that $s \equiv_{<} State$ and $t_i \equiv_{<} State$. Topmost rewrite theories provide a natural computation model for concurrent systems as shown in the following example.

Example 1. Let us consider a topmost rewrite theory $\mathscr{R} = (\Sigma, E \uplus B, R)$ that encodes a client-server communication protocol. The code snippet in Figure 1 shows a fragment of the theory signature Σ that includes the most relevant sorts and operators of the considered protocol specification.

Specifically, the signature Σ includes several operators and sorts that model the 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 $\mathtt{t}_1 \dots \mathtt{t}_n$, where, for the sake of simplicity, each $\mathtt{t}_\mathtt{i}$ is a term of sort `Symbol` in the alphabet `{a,b,c}`. We assume that the sort `Symbol` is a subsort of `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 that `C` wants to communicate with, `Q` is a message encoding a client 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 client status. Initially, the status is set to the initial default value `mt`, and it changes to `success` whenever a server acknowledges message reception. 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 request or a server response. System states are formalized as multisets $<\ \mathtt{t}_1\ \&\dots\&\ \mathtt{t}_\mathtt{m} >$ of clients, servers, and network packets via the associative and commutative operator `_&_` whose unity element is the constant `null`. System states have sort `State`.

The protocol dynamics is specified by the term rewriting system R in $\mathscr{R}$ that consists of the following three rewrite rules, where clients and servers agree on a shared key `K`.

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

More specifically, the rule `req` allows a client `C` to initiate a transmission request with a server `S` by sending a message `Q` that is encrypted by function `enc(Q,K)` using the client's key `K`. The rule `reply` lets the server `S` consume a client 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

```
--- sort specification

  sorts Nat Symbol Message Content State Packet
          Cli Serv Host CliName ServName Conf Status .

  subsort Symbol < Message .
  subsorts Packet Cli Serv < State .
  subsorts CliName ServName < Host .

--- operators for the client-server data structures

  op Srv-A Srv-B : -> ServName [ctor] .
  op Cli-A Cli-B : -> CliName [ctor] .
  op null : -> State [ctor] .
  op _&_ : State State -> State [ctor assoc comm id: null] .
  op _<-_ : Host Content -> Packet [ctor] .
  op {_,_} : Host Message -> Content [ctor] .
  op [_,_,_,_,_] : CliName ServName Message Nat Status -> Cli [ctor] .
  op [_,_] : ServName Nat -> Serv [ctor] .
  op <_> : State -> Conf [ctor] .
  op __ : Message Message -> Message [ctor assoc] .
  op a : -> Symbol [ctor] .
  op b : -> Symbol [ctor] .
  op c : -> Symbol [ctor] .
  op len : ->  Nat .
  op mt : -> Status [ctor] .
  op success : -> Status [ctor] .

--- operators for the Caeser chiper

  --- Symbol-to-Nat Nat-to-Symbol operators
  op toNat : Symbol -> Nat .
  op toSym : Nat -> Symbol .
  --- Encryption/Decryption operators
  op shift : Nat -> Nat .
  op unshift : Nat -> Nat .
  op en : Nat Nat -> Nat .
  op de : Nat Nat -> Nat .
  op enc : Message Nat -> Message .
  op dec : Message Nat -> Message .
```

Fig. 1. (Fragment of the) signature of the client-server communication protocol.

includes the decrypted request message. The rule `rec` successfully completes the data transmission between `C` and `S` whenever the server response packet `C <- {S,Q}` includes a message `Q` that 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 transmission succeeds when the client and server use the same key `K`.

```
var M : Message .
var X K : Nat .
var S : Symbol .

--- Function toNat(s) takes an alphabet symbol s as input and returns the
--- corresponding position in the alphabet.
eq toNat(a) = 0 [variant] .
eq toNat(b) = toNat(a) + s(0) [variant] .
eq toNat(c) = toNat(b) + s(0) [variant] .

--- Function toSym(n) takes a natural number n as input and returns the
--- corresponding alphabet symbol.
eq toSym(0)= a [variant] .
eq toSym(s(0)) = b [variant] .
eq toSym(s(s(0))) = c [variant] .

eq len = s(s(s(0))) --- Alphabet cardinality is equal to 3


--- Function shift(k) increments (modulo the alphabet cardinality)
--- the natural number k.
eq shift(X) = [ s(X) < len,s(X), 0 ] [variant] .

--- Function unshift(k) decrements (modulo the alphabet cardinality)
--- the natural number k.
eq unshift(0) = s(s(0)) [variant] .   --- (len - s(0))
eq unshift(s(X)) = X [variant] .

--- Function e(n,k) increments the natural number n by k units
--- (modulo the alphabet cardinality).
eq e(X,0) = X [variant] .
eq e(X,s(Y)) = e(shift(X),Y) [variant] .

--- Function d(n,k) decrements the natural number n by k units
--- (modulo the alphabet cardinality).
eq d(X,0) = X [variant] .
eq d(X,s(Y)) = d(unshift(X),Y) [variant] .

--- Function enc(m,k)  (resp. dec(m,k)) takes a message m and a
--- natural number k as input and returns the corresponding encrypted
--- (resp. decrypted) message using the Caesar cipher with key k.
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] .
```

Fig. 2. Equations of the equational theory encoding the *Caesar* cipher.

Encryption and decryption functionality is 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 ciphering where each symbol in the plaintext message is replaced by the symbol that appears `K` positions later in the alphabet (handled as the list `a,b,c`). The cipher is circular, i.e., it works modulo the cardinality of the alphabet. For instance, `enc(a b,s(0))` delivers `(b c)`, and `dec(a b,s(0))` yields the message `(c a)`. The equational theory $\mathcal{E}$ includes the equations[7] in Figure 2. In the specification, the equational attribute `variant` is used to identify the equations to be considered in the folding variant narrowing strategy, while any equations without the `variant` attribute are disregarded and are only considered for rewriting.

The complete Maude specification of the client-server communication protocol can be found in Appendix B.

2.1 Computing in Rewrite Theories

Given a rewrite theory $(\Sigma, E \uplus B, R)$, with $\mathcal{E} = (\Sigma, E \uplus B)$, the rewriting relation modulo $\mathcal{E}$ (in symbols, $\rightarrow_{R/\mathcal{E}}$) can be defined by lifting the usual rewrite relation on terms to the $E \uplus B$-congruence classes $[t]_{E\uplus B}$ on the term algebra $\mathcal{T}_\Sigma(\mathcal{X})$ that are induced by $=_\mathcal{E}$; in other words, $[t]_{E\uplus B}$ is the class of all terms that are equal to t *modulo* $E \uplus B$. This means $\rightarrow_{R/\mathcal{E}}$ is defined as $=_\mathcal{E} \circ \rightarrow_R \circ =_\mathcal{E}$.

A term t is called *$R/\mathcal{E}$-irreducible* iff there is no term u such that $t \rightarrow_{R/\mathcal{E}} u$. A substitution σ is $R/\mathcal{E}$-irreducible if, for every $x \in \mathcal{X}$, $x\sigma$ is $R/\mathcal{E}$-irreducible. We say that the relation $\rightarrow_{R/\mathcal{E}}$ is *terminating* if there is no infinite sequence $t_1 \rightarrow_{R/\mathcal{E}} t_2 \rightarrow_{R/\mathcal{E}} \cdots t_n \rightarrow_{R/\mathcal{E}} t_{n+1} \cdots$. We say that the relation $\rightarrow_{R/\mathcal{E}}$ is *confluent* if whenever $t \rightarrow^*_{R/\mathcal{E}} u$ and $t \rightarrow^*_{R/\mathcal{E}} v$, then u and v can be rewritten to some w up to $\mathcal{E}$-equality. A rewrite theory $(\Sigma, \mathcal{E}, R)$ is *convergent* if the rules R are sort-decreasing and the relation $\rightarrow_{R/\mathcal{E}}$ is confluent and terminating.

In a convergent order-sorted rewrite theory, for each term $t \in \mathcal{T}_\Sigma(\mathcal{X})$, there is a unique (up to $\mathcal{E}$-equivalence) $R/\mathcal{E}$-irreducible term t' that can be obtained by rewriting t to $R/\mathcal{E}$-irreducible or *normal* form, which is denoted by $t \rightarrow^!_{R/\mathcal{E}} t'$, or $t{\downarrow_{R/\mathcal{E}}}$ when t' is not relevant. For each $x \in Dom(\sigma)$, $\sigma{\downarrow_{R/\mathcal{E}}}$ is defined as $(\sigma{\downarrow_{R/\mathcal{E}}})(x) = \sigma(x){\downarrow_{R/\mathcal{E}}}$. A substitution σ is $R/\mathcal{E}$-irreducible (normalized) iff $x\sigma$ is so for each $x \in Dom(\sigma)$. For a set Q of terms, we denote by $Q{\downarrow_{R/\mathcal{E}}}$ the set of normal forms of the terms in Q.

Since $\mathcal{E}$-congruence classes can be infinite, $\rightarrow_{R/\mathcal{E}}$-reducibility is undecidable in general because any rewrite step $t \rightarrow_{R/\mathcal{E}} t'$ involves searching through the possibly infinite equivalence classes $[t]_{E\uplus B}$ and $[t']_{E\uplus B}$. Therefore, $R/\mathcal{E}$-rewriting is usually implemented by $R,\mathcal{E}$-rewriting. We define the relation $\rightarrow_{R,\mathcal{E}}$ on $\mathcal{T}_\Sigma(\mathcal{X})$ by $t \rightarrow_{p,R,\mathcal{E}} t'$ (or simply $t \rightarrow_{R,\mathcal{E}} t'$) iff there is a non-variable position $p \in Pos_\Sigma(t)$, a rule $\lambda \rightarrow \rho$ in R, and a substitution σ such that $t_{|p} =_\mathcal{E} \lambda\sigma$ and $t' = t[\rho\sigma]_p$. To ensure completeness of $R,\mathcal{E}$-rewriting w.r.t. $R/\mathcal{E}$-rewriting, we require $\mathcal{E}$-*coherence*: for any Σ-terms u, u', v if $u =_\mathcal{E} u'$ and $u \rightarrow_{R,\mathcal{E}} v$, then there exists a term v' such that $u' \rightarrow_{R,\mathcal{E}} v'$ and $v =_\mathcal{E} v'$. If $\mathcal{E}$-coherence holds for a set of rewrite rules R, we say that R is *$\mathcal{E}$-coherent*. Note that, assuming $\mathcal{E}$-matching is decidable, $\rightarrow_{R,\mathcal{E}}$ is decidable and notions such as confluence, termination, irreducible term, and normalized substitution are defined for $\rightarrow_{R,\mathcal{E}}$ straightforwardly [40].

[7] For the sake of simplicity, we omitted the definition of the operators `[_,_,_]`, `_<_`, and `_+_` that respectively implement the usual *if-then-else* construct, the *less-than* relation, and the associative and commutative addition over natural numbers.

2.2 Equational Theories as Rewrite Theories

Algebraic structures often involve axioms like associativity and/or commutativity of function symbols, which cannot be handled by ordinary term rewriting but are instead handled implicitly by working with congruence classes of terms. This is why the equation set of an equational theory is often decomposed into a disjoint union $E = E' \uplus B$, where B is a set of algebraic axioms (which are implicitly expressed in Maude as attributes of their corresponding operator using the `assoc`, `comm`, and `id:` keywords) that are used for B-matching, and E' is a set of equations that are implicitly oriented from left to right as a set $\vec{E}'$ of rewrite rules (and operationally used as simplification rules modulo B). By doing this, a (well-behaved) rewrite theory $(\Sigma, B, \vec{E}')$ can be defined, with $\vec{E}' = \{t \Rightarrow t' \mid t = t' \in E'\}$, which satisfies all of the conditions that we need.

This is formalized by the notion of *decomposition* $\vec{\mathscr{E}} = (\Sigma, B, \vec{E})$ of an equational theory $\mathscr{E} = (\Sigma, E \uplus B)$.

Definition 1 (Decomposition [30]**).** *Let $\mathscr{E} = (\Sigma, E \uplus B)$ be an order-sorted equational theory. We call $(\Sigma, B, \vec{E})$ a* decomposition *of $\mathscr{E}$ if $(\Sigma, B, \vec{E})$ is an order-sorted rewrite theory satisfying the following properties:*

1. *B is* regular*, i.e., for each $t = t'$ in B, we have $Var(t) = Var(t')$, and* linear*, i.e., for each $t = t'$ in B, each variable occurs only once in t and in t'.*
2. *B is* sort-preserving*, i.e., for each $t = t'$ in B and substitution σ, we have $t\sigma \in \mathscr{T}_{\Sigma,\mathsf{s}}(\mathscr{X})$ iff $t'\sigma \in \mathscr{T}_{\Sigma,\mathsf{s}}(\mathscr{X})$, for each sort s in the signature. Furthermore, for each equation $t = t'$ in B, all variables in $Var(t)$ and $Var(t')$ have a common top sort.*
3. *B has a finitary and complete unification algorithm, which implies that B-matching is decidable.*
4. *The rewrite rules in $\vec{E}$ are* convergent *(i.e., confluent, terminating, and* sort-decreasing*), and B-coherent.*

Normal forms $t \downarrow_{\vec{E},B}$ in a decomposition $(\Sigma, B, \vec{E})$ are also called *canonical forms*.

Note that the mild requirements listed in Definition 1 are satisfied by most of the equational theories that are used in practice. In particular, the axiom set of commonly occurring equational theories only includes combinations of associativity (A), commutativity (C) and identity (U) axioms, and all the three axioms are regular, linear and sort-preserving. Furthermore, there exist finitary and complete unification algorithms for modular axiom combinations such as ACU, AC, CU, C, and U. The remaining cases (namely, A and AU) are partially supported by Maude (hence by our framework) because unification modulo A and AU are generally infinitary, yet Maude generates a complete and finitary set of equational unifiers for many A and AU unification problems [26].

It is worth noting that the Maude system automatically provides B-coherence completion w.r.t. rewriting, for rules and equations, for any combination of associativity and/or commutativity and/or identity axioms. That is, the specified rules and equations are automatically completed with no need for user intervention. Note that, for narrowing derivations, B-coherence of rules and equations must be explicitly ensured by the user (see [41]). We often abuse notation and say that $(\Sigma, B, \vec{E}')$ is a decomposition of an order-sorted equational theory $(\Sigma, E \uplus B)$, where E' is the explicitly extended B-coherent completion of E.

The Maude interpreter implements rewriting modulo $E \uplus B$ by means of two much simpler relations than $\rightarrow_{R/\mathscr{E}}$ and $\rightarrow_{R,\mathscr{E}}$, namely $\rightarrow_{R,B}$ and $\rightarrow_{\vec{E},B}$, so that rules and (oriented) equations can be intermixed in the rewriting process by simply using an algorithm of matching modulo B.

Then, an $(R,E \uplus B)$-rewrite step $\rightarrow_{R,\vec{E}\uplus B}$ on a term t in the rewrite theory $\mathscr{R} = (\Sigma, E \uplus B, R)$ can be implemented, without loss of completeness, by applying the following rewrite strategy: (i) reduce t w.r.t. $\rightarrow_{\vec{E},B}$ to the canonical form $t\downarrow_{\vec{E},B}$; and (ii) rewrite $t\downarrow_{\vec{E},B}$ w.r.t. $\rightarrow_{R,B}$.

A rewrite sequence $t \rightarrow^*_{R,\vec{E}\uplus B} t'$ in the rewrite theory $\mathscr{R} = (\Sigma, E \uplus B, R)$ is then deployed as the (possibly infinite) rewrite sequence (with $t_0 = t$ and $t_n{\downarrow_{\vec{E},B}} = t'$)

$$t_0 \rightarrow^*_{\vec{E},B} t_0{\downarrow_{\vec{E},B}} \rightarrow_{R,B} t_1 \rightarrow^*_{\vec{E},B} t_1{\downarrow_{\vec{E},B}} \rightarrow_{R,B} \ldots \rightarrow_{R,B} t_n{\downarrow_{\vec{E},B}}$$

that interleaves $\rightarrow_{\vec{E},B}$ rewrite steps and $\rightarrow_{R,B}$ rewrite steps following the strategy mentioned above. Note that, following this strategy, after each rewrite step using $\rightarrow_{R,B}$, generally the resulting term t_i, $i = 1,\ldots,n$, is not in canonical normal form and is thus normalized before the subsequent rewrite step using $\rightarrow_{R,B}$ is performed. Also, in the precise strategy adopted by Maude, the last term of a finite computation is finally normalized before the result is delivered.

Example 2. Consider the rewrite theory $\mathscr{R} = (\Sigma, E \uplus B, R)$ in Example 1 together with the system state

$t_0 =$ `[Cli-A,Srv-A,a,shift(s(0)),mt] & [Srv-A,s(s(0))].`

Then, there exists the following one-step rewrite sequence

$$t_0 \rightarrow^*_{R,\vec{E}\uplus B} t_1{\downarrow_{\vec{E},B}} \quad \text{where}$$

$t_1{\downarrow_{\vec{E},B}} =$ `(Srv-A <- {Cli-A,c}) & [Srv-A,s(s(0))] & [Cli-A,Srv-A,a,s(s(0)),mt]`

The considered rewrite sequence specifies an initial communication request from client `Cli-A` to server `Srv-A` using the key `K=s(s(0))` and an initial message `a`. Maude implements $t_0 \rightarrow^*_{R,\vec{E}\uplus B} t_1{\downarrow_{\vec{E},B}}$ as the rewrite sequence

$$t_0 \rightarrow^+_{\vec{E},B} t_0{\downarrow_{\vec{E},B}} \rightarrow_{R,B} t_1 \rightarrow^+_{\vec{E},B} t_1{\downarrow_{\vec{E},B}}$$

where the `req` rule is applied. More specifically, the rewrite sequence $t_0 \rightarrow^+_{\vec{E},B} t_0{\downarrow_{\vec{E},B}}$ equationally simplifies the initial state t_0 into its canonical form $t_0{\downarrow_{\vec{E},B}}$ by completely evaluating the function call `shift(s(0))` into the term `s(s(0))`. Then the `req` rule is applied to $t_0{\downarrow_{\vec{E},B}}$ and a new term

$t_1 =$ `[Srv-A,s(s(0))] & (Srv-A <- {Cli-A,enc(a, s(s(0)))}) & [Cli-A,Srv-A,a,s(s(0)),mt]`

is yielded that is further simplified into the canonical form $t_1{\downarrow_{\vec{E},B}}$ by normalizing the function call `enc(a,s(s(0)))` with the oriented equations in $\vec{E}$.

2.3 Symbolic Computation in Rewrite Theories

Similarly to *rewriting modulo an equational theory* $\mathscr{E}$, where syntactic pattern-matching is replaced with matching modulo $\mathscr{E}$ (or $\mathscr{E}$-matching), in *narrowing modulo an equational theory* (i.e., narrowing with the rules in R modulo the equations and axioms in E), syntactic unification is replaced by *equational* unification (or $\mathscr{E}$-unification). More precisely, we define the *equational narrowing* relation $\leadsto_{R,\mathscr{E}}$ on $\mathscr{T}_\Sigma(\mathscr{X})$ by $t \leadsto_{\sigma,p,R,\mathscr{E}} t'$ (or simply $t \leadsto_{\sigma,R,\mathscr{E}} t'$ or even $t \leadsto_\sigma t'$) iff there is a non-variable position $p \in Pos_\Sigma(t)$, a rule $\lambda \Rightarrow \rho$ in R, and a substitution σ such that $t_{|p}\sigma =_{\mathscr{E}} \lambda\sigma$ and $t' = t[\rho]_p\sigma$. A term t is called $(R,\mathscr{E})$-*strongly irreducible* (also called a rigid normal form [11]) iff there is no term u such that $t \leadsto_{\sigma,p,R,\mathscr{E}} u$ for any position p, which amounts to say that no subterm of t unifies modulo B with the left-hand side of any equation of E.

In a topmost rewrite theory $\mathscr{R} = (\Sigma, E \uplus B, R)$, with $\mathscr{E} = (\Sigma, E \uplus B)$, $\leadsto_{R,\mathscr{E}}$ is implemented in Maude by means of a *three-layer* narrowing relation $\leadsto_{R,\vec{E} \uplus B}$ [22]:

1. An $(R, E \uplus B)$-narrowing step from s to t with a rule $l \Rightarrow r$ in R can be performed iff there is a $\mathscr{E}$-unifier θ of the Σ-equation $s = l$ such that $t = r\theta$.
2. In turn, each $\mathscr{E}$-unification problem $s =^?_{\mathscr{E}} l$ of Point 1 is solved by using *folding variant* narrowing in the equational theory $\mathscr{E}$ that computes a finite, minimal and complete set of $\mathscr{E}$-unifiers for $s = l$ under suitable requirements [31]. Following [44], this is done by *equationally* narrowing the term s =?= l (that encodes the unification problem $s =^?_{\mathscr{E}} l$) to an extra constant tt for denoting *success* in the rewrite theory $\mathscr{R}_0 = (\Sigma \cup \{=?=, tt\}, B, \vec{E} \cup \{\varepsilon\})$, where the extra[8] rewrite rule $\varepsilon = (X \ {=?=}\ X \Rightarrow tt)$ has been added to $\vec{E}$ in order to mimic unification of two terms (modulo B) as a narrowing step[9] that uses ε.
3. For each folding variant narrowing step using a rule in $\vec{E}$ modulo B in Point 2, B-unification algorithms are employed.

The search space of topmost narrowing computations in $(\Sigma, E \uplus B, R)$ (respectively, FV-narrowing computations in $(\Sigma, B, \vec{E})$) can be represented as a tree-like structure that we call topmost narrowing (respectively, FV-narrowing) tree.

Equational, $(R, E \uplus B)$-narrowing computations are natively supported by Maude version 3.0 for unconditional rewrite theories.

Example 3. Consider the (partial) specification of integer numbers defined by the equations $E = \{\mathsf{X} + 0 = \mathsf{X}, \mathsf{X} + \mathsf{s}(\mathsf{Y}) = \mathsf{s}(\mathsf{X} + \mathsf{Y}), \mathsf{p}(\mathsf{s}(\mathsf{X})) = \mathsf{X}, \mathsf{s}(\mathsf{p}(\mathsf{X})) = \mathsf{X}\}$, where variables X, Y are of sort `Int`, operators p and s respectively stand for the predecessor and successor functions, and B contains the commutativity axiom $\mathsf{X} + \mathsf{Y} = \mathsf{Y} + \mathsf{X}$. Also consider that the program signature Σ contains a binary state constructor operator $||\_,\_||$: `Int Int` $\rightarrow$ `State` for a new sort `State` that models a simple network of processes that are either performing a common task (denoted by the first component of the state) or have finished the task (denoted by the second component). The system state $t = ||\,\mathsf{s}(0), \mathsf{s}(0) + \mathsf{p}(0)\,||$ can be rewritten to $||\,0, \mathsf{s}(0)\,||$ (modulo the equations of E and the commutativity of $+$) using the following rule that specifies the system dynamics:

$$||\,\mathsf{A},\mathsf{B}\,|| \Rightarrow ||\,\mathsf{p}(\mathsf{A}), \mathsf{s}(\mathsf{B})\,||, \text{ where A and B are variables of sort } \texttt{Int} \qquad (1)$$

Also, a narrowing reachability goal from $||\,\mathsf{V} + \mathsf{V}, 0 + \mathsf{V}\,||$ to $||\,\mathsf{p}(0), \mathsf{s}(0)\,||$ succeeds (in one step) with computed answer substitution[10] $\{\mathsf{V} \mapsto 0\}$, which might signal a possible programming error in rule (1) since the number of processes in the first component of the state becomes negative.

[8] In an order-sorted setting, multiple equations are actually used to cover any possible sort in $\mathscr{R}$.

[9] For example, by using ε, the term $s(0) * 0$ =?= $U * s(V)$ FV-narrows to tt (modulo commutativity of $*$), and the computed narrowing substitution does coincide with the unifier modulo commutativity of the two argument terms, i.e., $\{U \mapsto 0, V \mapsto 0\}$.

[10] It is essentially calculated by first computing an $\mathscr{E}$-unifier σ of the input term $||\,\mathsf{V} + \mathsf{V}, 0 + \mathsf{V}\,||$ and the left-hand side $||\,\mathsf{A},\mathsf{B}\,||$ of rule (1), $\sigma = \{\mathsf{A}/(\mathsf{V} + \mathsf{V}), \mathsf{B}/\mathsf{V}\}$. Second, an $\mathscr{E}$-unifier σ' is computed between the instantiated right-hand side $||\,\mathsf{p}(\mathsf{V} + \mathsf{V}), \mathsf{s}(\mathsf{V})\,||$ and the target state $||\,\mathsf{p}(0), \mathsf{s}(0)\,||$, $\sigma' = \{\mathsf{V} \mapsto 0\}$. Third, the composition $\sigma\sigma' = \{\mathsf{A} \mapsto 0 + 0, \mathsf{B} \mapsto 0, \mathsf{V} \mapsto 0\}$ is simplified into $\{\mathsf{A} \mapsto 0, \mathsf{B} \mapsto 0, \mathsf{V} \mapsto 0\}$ and finally restricted to the variable V in the input term, yielding $\{\mathsf{V} \mapsto 0\}$.

3 Specialization of Rewrite Theories

In this section, we present the specialization procedure $\text{NPER}^{\mathscr{U}}_{\mathscr{A}}$, which allows a rewrite theory $\mathscr{R} = (\Sigma, E \uplus B, R)$ to be optimized by specializing the underlying equational theory $\mathscr{E} = (\Sigma, E \uplus B)$ with respect to the calls in the rewrite rules of R. The procedure $\text{NPER}^{\mathscr{U}}_{\mathscr{A}}$ extends the equational, narrowing-driven partial evaluation algorithm $\text{EQNPE}^{\mathscr{U}}_{\mathscr{A}}$ of [6], which applies to equational theories and is parametric on an unfolding operator $\mathscr{U}$ that is used to construct finite narrowing trees for any given expression and an abstraction operator $\mathscr{A}$ that guarantees global termination.

3.1 Partial Evaluation of Equational Theories

Given a convergent equational theory $\mathscr{E} = (\Sigma, E \uplus B)$ and a set Q of terms (henceforth called *specialized calls*), we define a transformation $\text{EQNPE}^{\mathscr{U}}_{\mathscr{A}}$ that derives a new equational theory $\mathscr{E}'$ which computes the same answers (and values) as $\mathscr{E}$ for any input term t that is a recursive instance (modulo B) of the specialized calls in Q. This means that all of the subterms of t (including itself) are a substitution instance of some term in Q. The transformation $\text{EQNPE}^{\mathscr{U}}_{\mathscr{A}}$ has two parameters, an *unfolding operator* $\mathscr{U}$ and an *abstraction operator* $\mathscr{A}$, whose precise meaning is clarified below.

The algorithm requires that the input equational theory $\mathscr{E}$ to be specialized is decomposed as a rewrite theory $\vec{\mathscr{E}} = (\Sigma, B, \vec{E})$, whose only equations are the equational axioms in B and where the equations in E are explicitly oriented from left to right as the set $\vec{E}$ of rewrite rules.

The transformation consists of iterating two consecutive actions:

(i) Symbolic execution (*Unfolding*). A finite, possibly partial folding variant narrowing tree for each input term t of Q is generated.[11] This is done by using the unfolding operator $\mathscr{U}(Q, \vec{\mathscr{E}})$ that determines when and how to stop the derivations in the FV-narrowing tree.

(ii) Search for regularities (*Abstraction*). In order to guarantee that all calls that may occur at runtime are *covered* by the specialization, every (sub-)term in any leaf of the tree must be *equationally closed* w.r.t. Q. This notion extends the classical PD closedness by:
   1) considering B-equivalence of terms;
   2) considering a natural partition of the signature as $\Sigma = \mathscr{D} \uplus \Omega$, where Ω are the *constructor* symbols, which are used to define the (irreducible) values of the theory (also called constructor terms), and $\mathscr{D}$ are the *defined* symbols, which are evaluated away by equational rewriting; and
   3) recursing over the term structure to handle nested function calls.

   Roughly speaking, a term u is equationally closed modulo B w.r.t. Q iff either: (i) it does not contain defined function symbols of $\mathscr{D}$, 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 $\bullet$ (i.e., $\bullet \in \mathscr{D}$) that does not obey any structural axioms, the term $t = a \bullet (Z \bullet a)$ is equationally closed w.r.t. $Q = \{a \bullet X, Y \bullet a\}$ or $\{X \bullet Y\}$, but it is not closed with Q being $\{a \bullet X\}$; however, it would be closed if $\bullet$ were commutative.

[11] For simplicity, we assume that Q is normalized w.r.t. the equational theory $\mathscr{E}$. If this were not the case, for each $t \in Q$ that is not in canonical form such that $t\downarrow_{\vec{E},B} = C(\overline{t_i})$, where $C(\ )$ is the (possibly empty) constructor context of $t\downarrow_{\vec{E},B}$ and $\overline{t_i}$ are the maximal calls in $t\downarrow_{\vec{E},B}$, we would replace t in Q with the normalized terms $\overline{t_i}$ and add a suitable "bridge" equation $t = C(\overline{t_i})$ to the resulting specialization.

Steps (i) and (ii) of the transformation are iterated as long as new terms are generated, and the considered abstraction operator $\mathscr{A}$ is used to guarantee that only finitely many expressions are evaluated, thus ensuring global termination.

For simplicity, the formulation of the EqNPE framework in [6] does not explicitly deal with subsort overloading of symbols, which could introduce subtle issues in the partial evaluation process, e.g., when a given function symbol is both a defined function symbol and a constructor operator. The new framework in this article does deal with subsort overloading via the following definition that naturally extends the equational closedness of [6] by introducing a least sort check to precisely identify constructor-rooted terms (i.e., terms whose top symbol is a constructor operator).

Definition 2 ((Extended) Equational Closedness). *Let $(\Sigma, B, \vec{E})$ be an equational theory decomposition and Q be a finite set of Σ-terms, i.e., terms that are built from Σ and a countably infinite set of variables $\mathscr{X}$. Assume the signature Σ splits into a set $\mathscr{D}$ of defined function symbols and a set Ω of constructor operators so that $\Sigma = \mathscr{D} \uplus \Omega$. We say that a Σ-term t is closed* modulo *B, or simply B-closed, (w.r.t. Q and Σ) if $closed_B(Q,t)$ holds, where the predicate $closed_B$ is defined as follows:*

$$closed_B(Q,t) \Leftrightarrow \begin{cases} true & if\ t \in \mathscr{X} \\ closed_B(Q,t_1) \wedge \ldots \wedge closed_B(Q,t_n) & if\ t = c(\overline{t_n : s_n}), \exists c : \overline{s_n} \to s \in \Sigma\ s.t. \\ & c \in \Omega,\ ls(t) = s,\ n \geq 0 \\ \bigwedge_{x \mapsto t' \in \theta} closed_B(Q,t') & if\ \exists q \in Q, \exists \theta\ such\ that \\ & root(t) = root(q) \in \mathscr{D}\ and \\ & q\theta =_B t \\ false & otherwise \end{cases}$$

A set T of terms is closed modulo B (w.r.t. Q and Σ) if $closed_B(Q,t)$ holds for each t in T. A set R of rules is closed modulo B (w.r.t. Q and Σ) if the set that can be formed by taking the right-hand sides of all of the rules in R is also closed modulo B. We often omit Σ when no confusion can arise.

The main difference of Definition 2 with respect to [6] is in the case when $t = c(\overline{t_n})$ because, due to subsort overloading, we might have an overloaded symbol in Σ with two different typings: a constructor typing and a defined typing. To cope with this, we need to search the very specific constructor declaration that matches the input term t; i.e., $c : \overline{s_n} \to s \in \Sigma,\ \ n \geq 0$, with $c \in \Omega$ and $ls(t) = s$. More precisely, we adopt the natural assumption that any constructor term is still constructor under instantiation (*preregular-below* condition in [50]). In other words, the preregular-below condition states that every overloaded symbol in Σ cannot have a defined typing that lies below any constructor typing for the same symbol in the sort poset.

Given a term t, its least-sort $ls(t)$ provides the information that is required to establish whether the top symbol of t is (or is not) a constructor operator and allows us to deal with non-free[12] constructor operators (whose behavior is defined through equations), which could not be handled by our previous framework in [6].

Example 4. Consider the following Maude program that encodes an equational theory with an overloaded operator c and an empty set of axioms $B = \emptyset$:

[12] A constructor operator $c \in \Omega$ is free in $\mathscr{E}$ if and only if, for every c-rooted term t, there is no c'-rooted constructor term t', with $c' \neq c$, such that $t \neq_B t'$ and $t =_{\mathscr{E}} t'$ (i.e., $t{\downarrow}_{\vec{E},B} = t'$). For the case when $B = \emptyset$, this is equivalent to say that free constructors only obey the strict equality axiom $c(X_1,..X_n) = c'(Y_1,..,Y_n)$ $\Leftrightarrow c = c'$ and $X_i = Y_i$, for $i = 1,..,n$.

```
fmod OVERLOAD is
 sorts A B .
 subsort A B < C .
 op a : -> A [ctor].
 op b : -> B [ctor].
 op c : A -> A [ctor] .
 op c : B -> B .
 eq c(X:B) = b .
endfm
```

The `[ctor]` attribute is the Maude syntax that is used to label constructor operators. Note that the program is preregular-below since the constructor typing `c : A -> A` for the overloaded operator `c` and its defined typing are incomparable in the sort poset. In particular, `c : B -> B` does not lie below `c : A -> A`.

The term `c(a)` is constructor and its least sort matches the declaration `op c : A -> A` whereas the term `c(b)` is not constructor and its least sort matches the declaration of the defined symbol `op c : B -> B`.

Also, for $Q = \emptyset$, $closed_{\emptyset}$(Q, `c(b)`) = `false`, since ls(`c(b)`) = B and the appropriate (defined) typing `op c : B -> B` is selected by $closed_{\emptyset}$ to establish that `c(b)` is not $\emptyset$-closed w.r.t. Q because it vacuously holds that there is no term $q \in Q$ that covers `c(b)`.

In contrast, our previous framework in [6] would have erroneously classified `c(b)` as $\emptyset$-closed w.r.t. Q because we did not support overloaded symbols that can be constructor or defined depending on their typing. Actually, the top symbol of `c(b)` would simply have been considered to be constructor (because of the constructor typing `op c : A -> A`) and thus the closedness check for `c(b)` w.r.t. Q would have succeeded since its argument, `b`, is also constructor.

Example 5. Consider the rewrite theory in Example 1 whose complete specification appears in Appendix B. The rewrite theory is trivially preregular-below since it does not include any overloaded symbol.

Given the set $\mathscr{L}$ of leaves in the FV-narrowing trees for the partially evaluated calls in Q, in order to properly add to Q the non-closed (sub-)terms occurring in the terms of $\mathscr{L}$, the abstraction operator $\mathscr{A}(Q,\mathscr{L},B)$ is applied, which yields a new set of terms which may need further evaluation. The abstraction operator $\mathscr{A}(Q,\mathscr{L},B)$ ensures that the resulting set of terms "covers" (modulo B) the calls previously specialized and that equational closedness modulo B is preserved throughout successive abstractions.

More formally, for the correctness of the equational partial evaluation, any instance of the generic abstraction operator $\mathscr{A}(Q,\mathscr{L},B)$ must agree with the following definition.

Definition 3 (Equational Abstraction [6]). *Given the finite set of terms T and the already evaluated set of terms Q, $\mathscr{A}(Q,\mathscr{L},B)$ returns a new set Q' such that:*

1. *if $v \in Q'$, then there exists $u \in (Q \cup T)$ and a renamed version v' of v, such that $u_{|p} =_B v'\theta$ for some position p and substitution θ*
2. *for all $t \in (Q \cup T)$, t is closed with respect to Q' modulo B.*

A concrete implementation of an abstract operator that meets the requirements of Definition 3 is shown in Section 5.1.

Note that the equational partial evaluation procedure 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 partial evaluation of the equations E. The partial evaluation of E basically consists of the set E' of *resultants* $t\sigma = t'$ that are associated with the derivations in the FV-narrowing tree from a root $t \in Q'$ to a leaf t' with computed substitution σ (i.e., the accumulated substitution along the narrowing derivation to the leaf). Note 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'. We assume the existence of a function $\textsc{GenTheory}(Q', (\Sigma, E \uplus B))$ that delivers the partially evaluated equational theory $\mathscr{E}' = (\Sigma', E' \uplus B')$ univocally determined by Q' and the original equational theory $\mathscr{E} = (\Sigma, E \uplus B)$, with $\Sigma' = \Sigma$ and $B' = B$. Formally,

$$\begin{aligned}\textsc{GenTheory}(Q', \mathscr{E}) = (\Sigma, \{t\sigma = t' \mid t \in Q', t' \in \mathscr{U}(Q', \vec{\mathscr{E}}),\ & t \text{ FV-narrows to } t' \\ & \text{with computed substitution } \sigma\} \uplus B).\end{aligned}$$

3.2 The $\text{NPER}^{\mathscr{U}}_{\mathscr{A}}$ Scheme for the Specialization of Rewrite Theories

The specialization of the rewrite theory $\mathscr{R} = (\Sigma, E \uplus B, R)$ is achieved by partially evaluating the hosted equational theory $\mathscr{E} = (\Sigma, E \uplus B)$ w.r.t. the rules of R, which is done by using the partial evaluation procedure $\text{EQNPE}^{\mathscr{U}}_{\mathscr{A}}$ of Section 3.1. By providing suitable unfolding and abstraction operators, different instances of the specialization scheme can be defined.

The $\text{NPER}^{\mathscr{U}}_{\mathscr{A}}$ procedure is outlined in Algorithm 1 and it consists of two phases.

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

```
Require:
    A rewrite theory $\mathscr{R} = (\Sigma, E \uplus B, R)$, an unfolding operator $\mathscr{U}$
 1: function $\text{NPER}^{\mathscr{U}}_{\mathscr{A}}(\mathscr{R})$
    Phase 1. Partial Evaluation
 2:     $R' \leftarrow \{(l\downarrow_{\vec{E},B}) \Rightarrow (r\downarrow_{\vec{E},B}) \mid l \Rightarrow r \in R\}$
 3:     $Q \leftarrow mcalls(R')$
 4:     $Q' \leftarrow \text{EQNPE}^{\mathscr{U}}_{\mathscr{A}}((\Sigma, E \uplus B), Q)$
 5:     $(\Sigma', E' \uplus B') \leftarrow \textsc{GenTheory}(Q', (\Sigma, E \uplus B))$
    Phase 2. Compression
 6:     $(\Sigma'', E'' \uplus B'', R'') \leftarrow \textsc{Compress}((\Sigma, E \uplus B, R'), (\Sigma', E' \uplus B'), Q')$
 7: return $(\Sigma'', E'' \uplus B'', R'')$
```

Phase 1 (Partial Evaluation). It applies the $\text{EQNPE}^{\mathscr{U}}_{\mathscr{A}}$ algorithm to specialize the equational theory $\mathscr{E} = (\Sigma, E \uplus B)$ w.r.t. a set Q of specialized calls that consists of all of the *maximal function calls* that appear in the $(\vec{E}, B)$-normalized version R' of the rewrite rules of R.

Given $\Sigma = (\mathscr{D} \uplus \Omega)$, a maximal function call in a term t is a subterm $t_{|w}$ of t, with $w \in Pos(t)$, such that (i) $root(t_{|w}) \in \mathscr{D}$, and (ii) there does not exist $w' \in Pos(t)$, such that $w' < w$ and $root(t_{|w'}) \in \mathscr{D}$. In other words, a *maximal* function call in a term t is any outermost subterm of t that is rooted by a defined function symbol of E. By $mcalls(R)$, we denote the set of all maximal calls in the rules of R.

Example 6. Let Σ be the signature $\{f,g,a,b,c\}$ where $\mathscr{D} = \{f,g\}$, $\Omega = \{a,b,c\}$, and g and c are associative and commutative operators. Then, the maximal function calls for the term $c(f(g(a,a)),c(g(g(b,b),a),a))$ are $f(g(a,a))$ and $g(g(b,b),a)$.

Example 7. Consider the set of rewrite rules R of the rewrite theory of Example 1 that specifies the dynamics of our client-server communication protocol. Then, $mcalls(R)$ ={`enc(Q,K)`, `dec(M,K)`}.

This phase produces the new set of specialized calls Q' from which the partial evaluation $\mathscr{E}' = (\Sigma', E' \uplus B')$ of $\mathscr{E}$ w.r.t. Q is univocally derived by GENTHEORY$(Q', (\Sigma, E \uplus B))$.

Phase 2 (Compression). It consists of a refactoring transformation that takes as input the rewrite theory $\mathscr{R}' = (\Sigma, E \uplus B, R')$, the computed partially evaluated theory $\mathscr{E}' = (\Sigma', E' \uplus B')$, and the final set of specialized calls Q' from which $\mathscr{E}'$ derives. Roughly speaking, the transformation computes a new, much more compact equational theory $\mathscr{E}'' = (\Sigma'', E'' \uplus B'')$ where unused symbols and unnecessary repetitions of variables are removed and equations of E' are simplified by recursively renaming all expressions that are Q'-closed modulo B by using an independent (i.e., overlap-free) renaming function that is derived from the set of specialized calls Q'.

Formally, an *independent renaming* ρ for Q' is a mapping from terms to terms that is defined as follows. For each t of sort s in Q' with $root(t) = f$, we define $\rho(t) = f_t(\overline{x_n : s_n})$, 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.

Example 8. Consider the rewrite theory in Example 1 together with the set of specialized calls

$$Q = \{\mathsf{dec(enc(M,s(s(0))))}, \mathsf{enc(enc(M,K1),K2)}\}.$$

An initial renaming ρ for Q is given by

$$\rho = \{\mathsf{dec(enc(M,s(s(0))))} \mapsto \mathsf{f0(M)}, \mathsf{enc(enc(M,K1),K2)} \mapsto \mathsf{f1(M,K1,K2)}\},$$

where `f0: Message -> Message` and `f1: Message Nat Nat -> Message` are new function symbols.

Compression is performed by the COMPRESS function given in Algorithm 2 that relies on the notion of *best fitting calls* (BFC), which is used in the renaming process for selecting the specialized calls from Q' that best cover a given call t. Formally, given t and a set U of terms, let $Anti_B(t,U) = \{u \in U \mid t =_B u\theta\}$ be the subset of U whose elements are more general (modulo B) than t (i.e., the anti-instances of t modulo B occurring in U). Then, we define the best fitting calls $BFC_B(t,U)$ for t in U w.r.t. B as the subset of minimally general elements of $Anti_B(t,U)$ (i.e., if $u \in BFC_B(t,U)$, then $BFC_B(t,U)$ does not contain any term that is strictly more general than u modulo B). In symbols,

$$BFC_B(t,U) = \{u \in Anti_B(t,U) \mid \nexists u' \in Anti_B(t,U) \text{ s.t. } u' <_B u\}$$

Note that $BFC_B(t,U)$ may contain more than one element, and hence multiple best fitting terms are possible for a given term t. Let us see an example.

Example 9. Consider a signature that contains an associative binary symbol $\oplus$ and the constant operators a, b and c. Let X and Y be variables. Let $t = a \oplus b \oplus c$ and $U = \{a \oplus X, Y \oplus b \oplus c\}$. Then, $BFC_B(t,U) = U$, because both $a \oplus X$ and $Y \oplus b \oplus c$ are anti-instances of t modulo associativity and neither $a \oplus X$ is more general than $Y \oplus b \oplus c$ modulo A nor *vice versa*.

Algorithm 2 Compression algorithm

```
Require:
    A rewrite theory $\mathscr{R}' = (\Sigma, E \uplus B, R')$, a partial evaluation $\mathscr{E}' = (\Sigma', E' \uplus B')$ of $(\Sigma, E \uplus B)$ w.r.t. a set of
    specialized calls $Q$.
1: function COMPRESS($\mathscr{R}, \mathscr{E}', Q$)
2:     Let $\rho$ 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
```

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

Essentially, the COMPRESS function recursively computes, by means of the function RN_ρ, a new equation set E'' by replacing each call t in E' by a call to the corresponding renamed function according to ρ and the best fitting calls for t in Q, i.e., $BFC_B(t,Q)$.[13]

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 and $BFC_B(t,Q)$. 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 $\mathscr{R}'' = (\Sigma'', E'' \uplus B'', R'')$ 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).

Also, observe that compression does not reduce the number of equations and rules of a rewrite theory, it just replaces (possibly) textually-large expressions in equations and rules with simpler ones that are obtained via the recursive renaming procedure. However, the effect of compression is much more striking than a mere reduction of the program size. This is because of two reasons. On the one hand, the set of axioms may be cut down whenever any operator that is equipped with some axioms and does not occur in the partially evaluated equations is taken

[13] Note that the function RN_ρ is actually non-deterministic since order-sorted anti-unification modulo axioms is not unitary, in contrast to untyped syntactic anti-unification. Hence, multiple (equally "best") renamings are possible for a term t under an independent renaming ρ. However, our implementation deterministically selects one element of $BFC_B(t,Q)$, thereby producing just one renaming for t.

out from the theory $\mathscr{E}'$, as shown in Example 17. On the other hand, potential overlaps among the specialized calls in the final set Q' are removed by applying the independent renaming ρ, and so is any spurious non-determinism that might have been introduced by the specialization process before compression. This is illustrated in the following example.

Example 10. Consider the equational definition

```
1 eq append(nil, L) = L [ variant ] .
2 eq append(X . L,L') =  X . append(L,L') [variant] .
3 eq append(append(nil, nil), L) = L [ variant ] .
4 eq append(append(nil, X . L), L') = X . append(L, L') [ variant ] .
5 eq append(append(X . L, L'), L'') = X . append(append(L, L'), L'') [ variant ] .
```

that can be obtained by partially evaluating the well-know `append` function for list concatenation w.r.t. the input call `append(append(L1:List,L2:List),L3:List)` that concatenates three lists by applying the `append` function twice.

The first two equations in the specification above reproduce the original definition of the `append` function where `nil` and `_._` are the usual list constructors, while the remaining three equations provide the intended specialization for the double append call. However, note that a given term `append(append(l1,l2),l3))` could be possibly narrowed by using equations 3, 4, and 5, but also using equations 1 and 2, which is certainly unintended, wastes the optimized function and has more indeterminism than the original definition.

The compression phase is able to eliminate this extra, spurious non-determinism by producing the following independent set of renamed equations where double append applications can now be reduced only by `f1` equations:

```
eq f0(nil, L) = L [ variant ] .
eq f0(X . L,L') =  X . f0(L,L') [variant] .
eq f1(nil, nil, L) = L~ [ variant ] .
eq f1(nil, X . L, L') = X . f0(L, L') [ variant ] .
eq f1(X . L, L', L'') = X . f1(L, L', L'') [ variant ] .
```

The following technical result holds for the specialization of the rewrite theories.

Theorem 1 (Preservation of executability conditions by NPER$^{\mathscr{U}}_{\mathscr{A}}(\mathscr{R})$**).** *Let* $\mathscr{R} = (\Sigma, E \uplus B, R)$ *be a topmost rewrite theory such that* $\mathscr{E} = (\Sigma, E \uplus B)$ *and* R *is* $\mathscr{E}$*-coherent. Let* $\vec{\mathscr{E}} = (\Sigma, B, \vec{E})$ *be a decomposition of* $\mathscr{E}$*, and let the left-hand sides of the rules in* R *be* $(\vec{E}, B)$-strongly irreducible. *Let* $\mathscr{U}$ *be an unfolding operator and let* $\mathscr{A}$ *be an abstract operator. Given the set* Q *of the maximal calls in the normalized rules of* R*, let* $\mathscr{R}' = (\Sigma', E' \uplus B', R')$ =NPER$^{\mathscr{U}}_{\mathscr{A}}(\mathscr{R})$ *be the specialization of* $\mathscr{R}$*, with* Q' =EQNPE$^{\mathscr{U}}_{\mathscr{A}}((\Sigma, E \uplus B), Q)$ *and* $\mathscr{E}' = (\Sigma', E' \uplus B')$ =GENTHEORY$(Q', (\Sigma, E \uplus B))$ *being the partial evaluation of* $\mathscr{E}$ *w.r.t* Q*. (under a given independent renaming* ρ *for* Q'*).*

Then, $\vec{\mathscr{E}}' = (\Sigma', B', \vec{E}')$ *is a decomposition of* $\mathscr{E}'$*,* R' *is* $\mathscr{E}'$*-coherent, and the left-hand sides of the rules in* R' *are* $(\vec{E}', B')$-strongly irreducible.

Now we are ready to establish the strong correctness of our specialization algorithm.

Theorem 2 (Strong correctness of NPER$^{\mathscr{U}}_{\mathscr{A}}(\mathscr{R})$**).** *Let* $\mathscr{R} = (\Sigma, E \uplus B, R)$ *be a topmost rewrite theory such that* $\mathscr{E} = (\Sigma, E \uplus B)$ *and* R *is* $\mathscr{E}$*-coherent. Let* $\vec{\mathscr{E}} = (\Sigma, B, \vec{E})$ *be a decomposition of* $\mathscr{E}$*, and let the left-hand sides of the rules in* R *be* $(\vec{E}, B)$-strongly irreducible. *Let* $\mathscr{U}$ *be an unfolding operator and let* $\mathscr{A}$ *be an abstract operator. Given the set* Q *of the maximal calls* *in the normalized rules of R, let* $\mathcal{R}' = (\Sigma', E' \uplus B', R')$ $=$NPER$^{\mathcal{U}}_{\mathcal{A}}(\mathcal{R})$ *be the specialization of* $\mathcal{R}$*, with* Q' $=$EQNPE$^{\mathcal{U}}_{\mathcal{A}}(\mathcal{E}, Q)$ *and* $\mathcal{E}' = (\Sigma', E' \uplus B')$ $=$GENTHEORY$(Q', (\Sigma, E \uplus B))$ *being the partial evaluation of* $\mathcal{E}$ *w.r.t Q. (under a given independent renaming* ρ *for* Q'*).*

Let $u \in \mathcal{T}_\Sigma(\mathcal{X})$ *be B-closed w.r.t.* Q' *and* Σ *and* $u' = RN_\rho(u) \in \mathcal{T}_{\Sigma'}(\mathcal{X})$*.*

1. $(u \rightarrow^*_{R,\vec{E} \uplus B} v)$ *if and only if* $(u' \rightarrow^*_{R',\vec{E}' \uplus B'} v')$*, with* $v' =_{B'} RN_\rho(v)$*.*
2. *If* $\mathcal{E}$ *satisfies the FVP, then for any* $(\vec{E}, B)$*-irreducible computed substitution* σ*,* $(u \rightsquigarrow^*_{\sigma,R,\vec{E} \uplus B} v)$ *if and only if* $(u' \rightsquigarrow^*_{\sigma',R',\vec{E}' \uplus B'} v')$*, with* $v' =_{B'} RN_\rho(v)$ *and* $\sigma' =_{B'} RN_\rho(\sigma)$*.*

Proofs of the (strong) correctness of the $NPER^{\mathcal{U}}_{\mathcal{A}}$ specialization algorithm and of the preservation of the executability conditions of the specialized theories are given in Appendix A.

4 FV-narrowing for Specializing Rewrite Theories

Given a rewrite theory $\mathcal{R} = (\Sigma, E \uplus B, R)$, with $\mathcal{E} = (\Sigma, B, \vec{E})$ being a decomposition of $(\Sigma, E \uplus B)$, the equational theory $\mathcal{E}$ in $\mathcal{R}$ may or may not meet the finite variant property (FVP). In this section, we formalize the notion of finite variant property and the related narrowing strategy that is called folded variant narrowing, which are the key ingredients to partially evaluate finite variant as well as non-finite vartiant equational theories.

Intuitively, given an equational theory $\mathcal{E} = (\Sigma, E \uplus B)$, the $(\vec{E}, B)$-variants (or simply variants) $(t\sigma{\downarrow_{\vec{E},B}}, \sigma)$ of t are the "irreducible patterns" $(t\sigma){\downarrow_{\vec{E},B}}$ to which t can be narrowed, with computed substitution σ, by applying the oriented equations $\vec{E}$ modulo B. For instance, there is an infinite number of variants for the term `(0 + Y:Int)` in the equation theory of Example 3; e.g., $(\texttt{Y:Int}, id)$, $(\texttt{0}, \{\texttt{Y:Int} \mapsto \texttt{0}\})$,$(\texttt{s(0)}, \{\texttt{Y:Int} \mapsto \texttt{s(0)}\})$, $(\texttt{s(Z:Int)}, \{\texttt{Y:Int} \mapsto \texttt{s(Z:Int)}\})$, $(\texttt{p(0)}, \{\texttt{Y:Int} \mapsto \texttt{p(0)}\})$, ...

A preorder relation of generalization between variants provides a notion of *most general variant* and also a notion of completeness of a set of variants. Formally, a variant (t, σ) is *more general* than a variant (t', σ') w.r.t. an equational theory $\mathcal{E}$ (in symbols, $(t, \sigma) \leq_{\mathcal{E}} (t', \sigma')$) iff $t \leq_{\mathcal{E}} t'$ and $\sigma \leq_{\mathcal{E}} \sigma'$. For instance, for the term $0 + \texttt{Y:Int}$, the most general variant is $(Y : \texttt{Int}, id)$ since any other variant can be obtained by equational instantiation.

Example 11. Consider the definition of the (associative and commutative) Boolean conjunction operator $\wedge$ given by $E = \{\texttt{X} \wedge \texttt{true} = \texttt{X}, \texttt{X} \wedge \texttt{false} = \texttt{false}\}$, where variable `X` belongs to sort `Bool` and constants `true` and `false` stand for the corresponding Boolean values. There are five most general variants modulo associativity and commutativity for the term $\texttt{X} \wedge \texttt{Y}$, which are: $\{(\texttt{X} \wedge \texttt{Y}, \texttt{id}), (\texttt{Y}, \{\texttt{X} \mapsto \texttt{true}\}), (\texttt{X}, \{\texttt{Y} \mapsto \texttt{true}\}), (\texttt{false}, \{\texttt{X} \mapsto \texttt{false}\}), (\texttt{false}, \{\texttt{Y} \mapsto \texttt{false}\})\}$.

An equational theory has the *finite variant property* (FVP) (or it is called a *finite variant theory*) iff there is a finite and complete set of most general variants for each term. It is generally undecidable whether an equational theory has the FVP [20]; a semi-decision procedure is given in [39] (and implemented in [8]) that works well in practice. The procedure in [39] works by computing the variants of all flat terms $f(X_1, \ldots, X_n)$ for any n-ary operator f in the theory and pairwise-distinct variables $X_1, \ldots, X_n$ (of the corresponding sort); the theory does have the FVP iff there is a finite number of most general variants for every such term.

For example, the theory of Example 11 satisfies the FVP since the flat term `(X and Y)` has only five most general variants. In contrast, the equational theory of Example 1 does not have the finite variant property; for instance, the term `d(X,Y)` has an infinite number of most general variants (`X`, $\{$`Y`$\mapsto$`0`$\}$), (`unshift(X)`, $\{$`Y`$\mapsto$`s(0)`$\}$), …, (`unshift`^{k}`(X)`, $\{$`Y`$\mapsto$ `s`^{k}`(0)`$\}$).

In [31], folding variant narrowing was proved to be complete, minimal, and finitary for variant generation and variant $E \uplus B$-unification w.r.t. $(\vec{E},B)$-normalized substitutions, provided that the theory has the FVP.

FV-narrowing derivations correspond to sequences $t_0 \leadsto_{\sigma_0,\vec{e}_0,B} t_1 \leadsto_{\sigma_1,\vec{e}_1,B} \cdots \leadsto_{\sigma_n,\vec{e}_{n-1},B} t_n$, where $t \leadsto_{\sigma,\vec{e},B} t'$ (or simply $t \leadsto_{\sigma} t'$ when no confusion can arise) denotes a transition (modulo the axioms in B) from term t to t' via the *variant equation* e (i.e., an oriented equation $\vec{e}$ that is enabled to be used for FV-narrowing thanks to the attribute `variant`) using the *equational unifier* σ. The composition $\sigma_0\sigma_1\sigma_{n-1}$ of all the unifiers along a narrowing sequence leading to t_n (restricted to the variables of t_0) is the computed substitution of this sequence. By notation $t \leadsto_{\sigma,\vec{E},B} t'$ (or also $t \leadsto_{\vec{E},B} t'$), we denote a FV-narrowing step which is performed using some oriented equation in $\vec{E}$. Also, notation $t \leadsto^n_{\sigma,\vec{E},B} t'$ (or simply $t \leadsto^n_{\vec{E},B} t'$) denotes a FV-narrowing derivation of exactly n FV-narrowing steps. The set of all FV-narrowing computations for a term t in $\mathscr{E}$ can be represented as a tree-like structure, denoted by $VN^{\circlearrowleft}_{\vec{\mathscr{E}}}(t)$, which we call the FV-narrowing tree of t in $\mathscr{E}$.

Assuming that the initial term t is normalized, each (variant narrowing) step $t \leadsto_{\sigma,\vec{e},B} t'$ is followed by the simplification of the term into its normal form by using all of the equations in the theory, which may include not only the variant equations in the theory but also (non-variant) equations (i.e., equations without the `variant` attribute). More precisely, given a rewrite theory $\mathscr{R} = (\Sigma, E \uplus G \uplus B, R)$, where B is the axiom set, E is the set of variant equations, and G is the set of non-variant equations, Maude does not perform narrowing with R modulo $E \uplus G \uplus B$, but only modulo $E \uplus B$, while equational simplification is carried out modulo the whole equational set $E \uplus G \uplus B$. This gives a more flexible narrowing relation for rewrite rules, which is particularly useful when only the equational theory fragment $(\Sigma, E \cup B)$ has the FVP (while equations in G would break it) so that variant $E \uplus B$-unification is finitary, whereas variant $E \uplus G \uplus B$-unification would be infinitary and undecidable. Therefore, in this scenario, each narrowing step with a rewrite rule r of the form $t \leadsto_{\sigma,R,\vec{E}\uplus B} t'$ is followed by simplification using the rewrite relation $\rightarrow^!_{\vec{G}\uplus\vec{E}\uplus B}$, i.e., the combined relation $(\leadsto_{\sigma,p,R,\vec{E}\uplus B};\rightarrow^!_{\vec{G}\uplus\vec{E},B})$ is defined as $t \leadsto_{\sigma,p,R,\vec{E}\uplus B};\rightarrow^!_{\vec{G}\uplus\vec{E},B} t''$ iff $t \leadsto_{\sigma,p,R,\vec{E}\uplus B} t'$, $t' \rightarrow^*_{\vec{G}\uplus\vec{E},B} t''$, and $t'' = t'\downarrow_{\vec{G}\uplus\vec{E},B}$.

An important number of verification tools and techniques rely on narrowing-based variant generation: for example, protocol analyzers, proofs of coherence and local confluence, termination provers, variant-based satisfiability checkers, and different applications of symbolic reachability analyses (references can be found in [8]).

5 Instantiating the Specialization Scheme for FVP and non-FVP Theories

Recall that the parameterized NPER$^{\mathscr{U}}_{\mathscr{A}}$ algorithm of Section 3.2 relies on two generic operators: an unfolding operator $\mathscr{U}$ that defines the unfolding rule used to determine when and how to terminate the construction of the narrowing trees; and an abstraction operator $\mathscr{A}$ that is used to guarantee that the set of terms obtained during partial evaluation (i.e., the set of deployed narrowing trees) is kept finite and progressively covers (modulo B) all of the specialized calls. The instantiation of the scheme requires particularizing these two parameters in order to specify a terminating, correct, and complete partial evaluation for $\mathscr{E}$. We provide two different implementations for the unfolding operator $\mathscr{U}$, namely, $\mathscr{U}_{\mathit{fvp}}$ and $\mathscr{U}_{\overline{\mathit{fvp}}}$. Both implementations exploit the folding variant narrowing strategy outlined in Section 4. Furthermore, we will resort to a single concrete definition of the abstraction operator $\mathscr{A}$ that works for both cases and is based on an equational generalization algorithm.

5.1 Abstraction Operator via Least General Generalizations

In general, there is no guarantee that the leaves $\mathscr{L}$ of the FV-narrowing trees are B-closed w.r.t. the specialized calls in Q. Indeed, the chosen unfolding operator $\mathscr{U}$ might deliver uncovered function calls that should be subsequently considered for specialization, while avoiding the set of partially evaluated calls from growing infinitely. In the following we introduce an abstraction operator $\mathscr{A}_{Elgg}(Q,\mathscr{L},B)$ that returns a set Q' of specialized calls that abstracts the set $Q \cup \mathscr{L}$ by using the generalization process formalized in [6] that ensures that Q' is B-closed w.r.t. $Q \cup \mathscr{L}$.

The abstraction operator $\mathscr{A}_{Elgg}(Q,\mathscr{L},B)$ relies on an *equational order sorted extension* of the pure, syntactical least general generalization algorithm [9] so that not too much precision is lost despite the abstraction. Roughly speaking, the syntactic generalization problem for two or more expressions, in a pure syntactic and untyped setting, means finding their *least general generalization* (lgg), i.e., the least general expression t such that all of the given expressions are instances of t under appropriate substitutions. For instance, the expression *sibling(X,Y)* is a generalizer of both *sibling(john,sam)* and *sibling(tom,sam)*, but their least general generalizer is *sibling(X,sam)*.

In [9], the notion of least general generalization is extended to the order-sorted modulo axioms setting, where function symbols can obey any combination of associativity, commutativity, and identity axioms (including the empty set of such axioms). For instance, the least general generalizer of *sibling(sam,john)* and *sibling(tom,sam)* is still *sibling(X,sam)*, when *sibling* is a commutative symbol. In general, there is no unique lgg in the framework of [9], due to both the order-sortedness and to the equational axioms. Nonetheless, for the case of modular combinations of associativity and commutativity axioms, there is always a finite, minimal, and complete set of equational lggs (E-lggs) so that any other generalizer has at least one of them as a B-instance. This result is fully extended in [14] to any modular combinations of associativity, commutativity, and identity axioms.

Formally, given an order-sorted signature Σ and a set of algebraic axioms B, a *generalization* modulo B of the nonempty set of terms $\{t_1,\ldots,t_n\}$ is a pair $\langle t,\Theta\rangle$, where $\Theta = \{\theta_1,\ldots,\theta_n\}$ is a set of substitutions, such that, for all $i = 1,\ldots,n$, $t\theta_i =_B t_i$. The pair $\langle t,\Theta\rangle$ is the *least general generalization* modulo B of a set of terms S, written $lgg_B(S)$, if (1) $\langle t,\Theta\rangle$ is a generalization of S and (2) for every other generalization $\langle t',\Theta'\rangle$ of S, t' is more general than t modulo B.

Let us introduce the notion of *best matching set* that is aimed at avoiding loss of specialization due to generalization. This notion is a proper, equational extension of [1] that we use to select the more appropriate terms in a given set U that cover a new call t. Roughly speaking, we determine the best matching set for t in a set U of terms w.r.t. B, $BMS_B(U,t)$, as follows: for each u_i in U, we compute the set $W_i = lgg_B(\{u_i,t\})$ and select the subset M of minimal upper bounds of the union $W = \bigcup_i W_i$. Then, the term u_k belongs to $BMS_B(U,t)$ if at least one element in the corresponding W_k belongs to M.

Let us introduce a simple motivating example, with $B = \emptyset$.

Example 12. Let $Q = \{f(g(a)), f(g(b)), f(a)\}$ and $t = f(g(d(a,b)))$. To compute the best matching set for t in Q, we first consider the set

$$\begin{aligned}W &= lgg(\{f(g(a)), f(g(d(a,b)))\}) \cup lgg(\{f(g(b)), f(g(d(a,b)))\}) \cup lgg(\{f(a), f(g(d(a,b)))\})\\ &= \{f(g(x)), f(g(y)), f(z)\}\end{aligned}$$

Now, the minimally general elements of W are $f(g(x))$ and $f(g(y))$, and thus we have $BMS_B(Q,t) = \{f(g(a)), f(g(b))\}$.

Definition 4 (Best Matching Set modulo B). *Let $\mathscr{E} = (\Sigma, E \uplus B)$ be an order-sorted equational theory. Let $U = \{u_1, \ldots, u_n\}$ be a set of terms and t be a term. Given the decomposition $(\Sigma, B, \vec{E})$ of $(\Sigma, E \uplus B)$, consider the sets of terms $W_i = \{w \mid \langle w, \{\theta_1, \theta_2\}\rangle \in lgg_B(\{u_i, t\})\}$, for $i = 1,..,n$, and $W = \bigcup_{i=1}^{n} W_i$. The* best matching set *$BMS_B(U,t)$ for t in U modulo B is the set of those terms $u_k \in U$ such that the corresponding W_k contains a* minimally general *element w of W under $\leq_B$, i.e., there is no different element w' in W (modulo the relation $\simeq_B$ induced by $\leq_B$) such that $w <_B w'$.*

The following example illustrates the above definition.

Example 13. Let $t = g(1) \otimes 1 \otimes g(Y)$, $U \equiv \{1 \otimes g(X), X \otimes g(1), X \otimes Y\}$, and consider B to consist of the associativity and commutativity axioms for the function symbol $\otimes$. To compute the best matching set for t in U, we first compute the sets of lgg_B's of t with each of the terms in U:

$$\begin{aligned}&W_1 = lgg_{AC}(\{g(1) \otimes 1 \otimes g(Y), 1 \otimes g(X)\}) = \{\langle Z \otimes 1, \{\{Z/g(1) \otimes g(Y)\}, \{Z/g(X)\}\}\rangle,\\ &\qquad \langle Z \otimes g(W), \{\{Z/1 \otimes g(1), W/Y\}, \{Z/1, W/X\}\}\rangle\}\\ &W_2 = lgg_{AC}(\{g(1) \otimes 1 \otimes g(Y), X \otimes g(1)\}) = \{\langle Z \otimes g(1), \{\{Z/1 \otimes g(Y)\}, \{Z/X\}\}\rangle\}\\ &W_3 = lgg_{AC}(\{g(1) \otimes 1 \otimes g(Y), X \otimes Y\}) = \{\langle Z \otimes W, \{\{Z/1, W/g(1) \otimes g(Y)\}, \{Z/X, W/Y\}\}\rangle\}\end{aligned}$$

Now, the set M of minimal upper bounds of the set $W_1 \cup W_2 \cup W_3$ is $M = \{\langle Z \otimes 1, \{\{Z/g(1) \otimes g(Y)\}, \{Z/g(X)\}\}\rangle, \langle Z \otimes g(1), \{\{Z/1 \otimes g(Y)\}, \{Z/X\}\}\rangle\}$ and thus we have: $BMS_{AC}(U,t) = \{1 \otimes g(X), X \otimes g(1)\}$.

Now, we are ready to instantiate the abstraction parameter $\mathscr{A}$ of our specialization procedure with the following function $\mathscr{A}_{Elgg}(Q,T,B)$ that relies on the notions of best matching set modulo B and equational least general generalization. Given the current set Q of already specialized calls, in order to augment Q with a new set T of terms, the best matching set is used when selecting the most appropriate terms of Q to be used for generalizing T, in the sense of providing appropriate least general generalizations.

Definition 5 (Abstraction operator). *Let $\mathscr{E} = (\Sigma, E \uplus B)$ be an order-sorted equational theory, with $\Sigma = \mathscr{D} \uplus \Omega$. Let $U = \{u_1, \ldots, u_n\}$ be a set of terms and t be a term. Given the decomposition $(\Sigma, B, \vec{E})$ of $(\Sigma, E \uplus B)$, Let Q, T be two sets of terms. We define $\mathscr{A}_{Elgg}(Q,T,B) = abs^{\breve{\trianglelefteq}_B}(Q,T)$, where:*

$$\begin{cases} abs^{\breve{\trianglelefteq}_B}(\ldots abs^{\breve{\trianglelefteq}_B}(Q, \{t_1\}), \ldots, \{t_n\}) & \text{if } T = \{t_1, \ldots, t_n\}, n > 1\\ Q & \text{if } T = \emptyset \text{ or } T = \{X\}, \text{with } X \in \mathscr{X}\\ abs^{\breve{\trianglelefteq}_B}(Q, \{t_1, \ldots, t_n\}) & \text{if } T = \{c(\overline{t_n : s_n})\} \text{ and } \exists c : \overline{s_n} \to s \in \Sigma \text{ s.t. } c \in \Omega, ls(t) = s, n \geq 0\\ generalize_B(Q, Q', t) & \text{if } T = \{t\}, \text{otherwise}\end{cases}$$

where $Q' = \{t' \in Q \mid root(t) = root(t') \text{ and } t' \trianglelefteq_B t\}$. *The function* generalize *is defined as follows:*

$$\begin{array}{lll} generalize_B(Q, \emptyset, t) & = & Q \cup \{t\} \\ generalize_B(Q, Q', t) & = & Q \text{ if } t \text{ is } B\text{-closed w.r.t. } Q \text{ and } \Sigma \\ generalize_B(Q, Q', t) & = & abs^{\check{\trianglelefteq}_B}(Q \setminus BMS_B(Q', t), Q''{\downarrow_{\vec{E},B}}) \quad (otherwise) \end{array}$$

where $Q'' = \{l \mid q \in BMS_B(Q', t), \langle w, \{\theta_1, \theta_2\}\rangle \in lgg_B(\{q, t\}), x \in Dom(\theta_1) \cup Dom(\theta_2),\ l \in \{w, x\theta_1, x\theta_2\}\}$.

It is worth noting that Definition 5 slightly generalizes the original formulation in [6] in order to fully deal with subsort overloading (including the overloading of constructor symbols). The new formalization of $\mathscr{A}_{Elgg}(Q,T,B)$ searches for the very specific constructor declaration that matches the input term t, similarly to the extended notion of equational closedness of Definition 2; i.e., $c : \overline{s_n} \to s \in \Sigma,\ n \geq 0$, with $c \in \Omega$ and $ls(t) = s$. The following example illustrates the improved specialization power that we achieve by this extension.

Example 14. The Maude functional module `OS-NAT/2` provides a specification of natural numbers modulo 2.

```
fmod OS-NAT/2 is
  sorts Nat Zero One .
  subsort Zero One < Nat .
  op 0 : -> Zero [ctor] .
  op s : Zero -> One [ctor] .
  op s : Nat -> Nat .
  eq s(s(0)) = 0 [variant] .
endfm
```

This module introduces two subsorts, namely, `Zero` and `One`, whose aim is to respectively type the two values `0` and `s(0)`. Furthermore, operator overloading is used to provide two versions of the successor operator `s` so that for one typing is a constructor symbol and for another typing is a defined symbol. More specifically, `s` : `Zero` $\to$ `One` specifies a constructor symbol that builds the canonical form `s(0)`, while `s` : `Nat` $\to$ `Nat` is a defined symbol that is used to simplify any natural number $\mathtt{s}^{\mathtt{k}}(0)$, with $\mathtt{k} \geq 2$, to either the value `0` or `s(0)` via the equation `s(s(0)) = 0`.

The term `s(0)` is constructor and matches the declaration `op s : Zero -> One` whereas the term `s(s(0))` is not and it matches the declaration of the defined symbol `op s : Nat -> Nat`.

Also, for $Q = \emptyset$, $\mathscr{L} = \{\mathtt{s(s(0))}\}$, and axiom set $B = \emptyset$, we have $\mathscr{A}_{Elgg}(Q, \mathscr{L}, B) = \{\mathtt{s(s(0))}\}$ since $ls($`s(s(0))`$)$ = `Nat`. This allows the appropriate (defined) typing `op s : Nat -> Nat` to be considered and the uncovered call `s(s(0))` is added to Q.

On the contrary, our previous framework in [6] would have erroneously classified `s(s(0))` as a constructor term and, as a result, the uncovered call `s(s(0))` could not have been specialized.

Theorem 3. *The operator* $\mathscr{A}_{Elgg}(Q, \mathscr{L}, B)$ *terminates and is an abstraction operator in the sense of Definition 3.*

5.2 Unfolding operators

Let us provide two possible implementations of the unfolding operator $\mathscr{U}$ that are respectively able to deal with: (a) equational theories that do not satisfy the FVP; and (b) equational theories that satisfy the FVP. Since $(\Sigma, B, \vec{E})$ is a decomposition of $(\Sigma, E \uplus B)$, both implementations adopt the folding variant narrowing strategy to build the narrowing trees that are needed to specialize the input theory.

(a) When $\mathcal{E}$ does not meet the finite variant property, folding variant narrowing may lead to the creation of an infinite FV-narrowing tree for some specialized calls in Q. In this case, the unfolding rule must implement a form of local control that stops the expansion of infinite derivations in the FV-narrowing tree. A solution to this problem has already been provided in [6] by means of an unfolding operator that computes a finite (possibly partial) FV-narrowing tree fragment for every specialized call t in Q. Narrowing derivations in the tree are stopped when no further FV-narrowing step can be performed or potential non-termination is detected by applying a subsumption check at each FV-narrowing step. The subsumption check is based on an *equational order-sorted* extension $\breve{\trianglelefteq}_B$ [7] of the classical homeomorphic embedding relation $\trianglelefteq$ that is commonly used to ensure termination of symbolic methods and program optimization techniques.
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 $t \triangleright t'$, if t' can be obtained from t by deleting some parts, e.g., $s(s(X+Y)*(s(X)+Y))$ embeds $s(Y*(X+Y)))$. Embedding relations have become very popular to ensure termination of *symbolic* transformations because, provided the signature is finite, for every infinite sequence of terms $t_1, t_2, \ldots$, there exists $i < j$ such that $t_i \trianglelefteq t_j$. In other words, the embedding relation is a well-quasi order (wqo) [35]. Therefore, 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 in t_{n+1}. Otherwise, t_{n+1} is added to the sequence and the computation can proceed.
The $\mathcal{U}_{\overline{fvp}}(Q, \vec{\mathcal{E}})$ operator implements an unfolding rule that is based on the homeomorphic relation $\breve{\trianglelefteq}_B$, whose full formalization relies on the following auxiliary notion.
We say that a FV-narrowing derivation D is *admissible w.r.t.* $\breve{\trianglelefteq}_B$ if and only if it does not contain a pair of comparable narrowing redexes (i.e., rooted by the same operation symbol) s and t, where s precedes t in D, such that $s \breve{\trianglelefteq}_B t$.

Example 15. Consider the equational theory $\mathcal{E} = (\Sigma, E \uplus B)$ that encodes the *Caesar* cipher of Example 1 and the input term `dec(M:Message,s(0))` that represents the decryption of a generic message `M` w.r.t. the key `s(0)`. Then, we have the following FV-narrowing derivations for `dec(M:Message,s(0))` in $\mathcal{E}$:

$$\mathtt{dec(M:Message,s(0))} \leadsto_{\sigma',\vec{E},B} \mathtt{toSym(unshift(toNat(M':Symbol)))} \leadsto_{\sigma'',\vec{E},B} \mathtt{a}$$

with $\sigma' = \{\mathtt{M:Message/M':Symbol}\}$ and $\sigma'' = \{\mathtt{M'/b}\}$.

$$\mathtt{dec(M:Message,s(0))} \leadsto_{\sigma'',\vec{E},B} \mathtt{toSym(unshift(toNat(S':Symbol)))\ dec(M':Message,s(0))}$$

with $\sigma'' = \{\mathtt{M:Message/(S':Symbol\ M':Message)}\}$. While the former is an admissible derivations, the latter is not, since there exists the trivial embedding `dec(M:Message,s(0))` $\breve{\trianglelefteq}_B$ `dec(M′:Message, s(0))`.

Definition 6 (Unfolding function). *Given the equational theory $\mathcal{E} = (\Sigma, E \uplus B)$ with a decomposition $\vec{\mathcal{E}} = (\Sigma, B, \vec{E})$, and a term t_0 to be specialized in $\mathcal{E}$, we define Unfold(t_0,$\vec{\mathcal{E}}$) as the set of terms given by*

$$\begin{aligned} \mathit{Unfold}^{\breve{\trianglelefteq}_B}(t_0, \vec{\mathcal{E}}) = \{t_n \mid\ & t_0 \leadsto^{n}_{\vec{E},B} t_n \in VN^{\circlearrowleft}_{\vec{\mathcal{E}}}(t_0), \\ & t_0 \leadsto^{n-1}_{\vec{E},B} t_{n-1} \textit{ is admissible w.r.t. } \breve{\trianglelefteq}_B \textit{ and} \\ & \textit{either } \nexists w : t_0 \leadsto^{n}_{\vec{E},B} t_n \leadsto_{\vec{E},B} w \in VN^{\circlearrowleft}_{\vec{\mathcal{E}}}(t_0) \\ & \textit{or } t_0 \leadsto^{n}_{\vec{E},B} t_n \textit{ is not admissible w.r.t. } \breve{\trianglelefteq}_B.\} \end{aligned}$$

Given a set Q of terms, we also define $\mathcal{U}_{\overline{fvp}}(Q, \vec{\mathcal{E}}) = \bigcup_{t \in Q} \mathit{Unfold}^{\breve{\trianglelefteq}_B}(t, \mathcal{R})$.

Note that $\mathcal{U}_{\overline{fvp}}(Q, \vec{\mathcal{E}})$ of Definition 6 computes a finite (possibly partial) folding variant narrowing tree in $\vec{\mathcal{E}}$ for each term t in Q and returns the set of the (normalized) leaves of the trees. Derivations are stopped when there is no further folding variant narrowing steps or the embedding whistle blows.

Example 16. Consider a specific instance of the rewrite theory of Example 1, where servers and clients use 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 FV-narrowing trees associated with encryption and decryption functionality may be infinite since $\mathcal{E}$ does not have the FVP, as shown in Section 4. For instance, terms of the form $(\mathtt{t_1} \ \ldots \ \mathtt{t_n} \ \mathtt{enc(M', s(s(0)))})$ derive from $\mathtt{enc(M, s(s(0))}$ by FV-narrowing, where $\mathtt{enc(M', s(s(0)))}$ can be further narrowed to unravel an unlimited sequence of identical terms modulo renaming. Nonetheless, homeomorphic embedding detects this non-terminating behavior since $\mathtt{enc(M', s(s(0))}$ embeds $\mathtt{enc(M, s(s(0))}$.
By using the unfolding operator $\mathcal{U}_{\overline{fvp}}$, the first phase of the NPER$^{\mathcal{U}}_{\mathcal{A}}(\mathcal{R})$ algorithm, with $\mathcal{U} = \mathcal{U}_{\overline{fvp}}$ and $\mathcal{A} = \mathcal{A}_{Elgg}$, computes the initial set $\mathtt{Q} = \{\mathtt{enc(M, s(s(0))}, \mathtt{dec(M, s(s(0)))}\}$ consisting of the (normalized) maximal function calls of $\mathcal{R}$. Then, the equational theory $\mathcal{E}$ is partially evaluated w.r.t. Q by EQNPE$^{\mathcal{U}}_{\mathcal{A}}$, using the tandem $\mathcal{U}_{\overline{fvp}}/\mathcal{A}_{Elgg}$. During the partial evaluation process, $\mathcal{U}_{\overline{fvp}}$ only unravels finite fragments of the FV-narrowing trees that are rooted by the specialized calls, thereby yielding the partial evaluation $\mathcal{E}'$ of $\mathcal{E}$ in Figure 3.
After the second phase of the algorithm, Presto produces the compressed equational theory $\mathcal{E}''$ of Figure 4 by computing the following renaming for the specialized calls:

```
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)
```

which eliminates prolix nested calls and redundant arguments in $\mathcal{E}''$ computations.
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. This transformation

```
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] .
```

Fig. 3. Specialization algorithm, Phase 1: Partial evaluation of $\mathscr{E}$ w.r.t. Q.

power cannot be achieved by existing, functional, logic or functional logic partial evaluators. Encryption (resp., decryption) in $\mathscr{E}''$ is now the direct mapping f0 (resp., f1) that associates messages to their corresponding encrypted (resp. decrypted) counterparts, avoiding a huge amount of computation in the profuse domain of natural numbers. Finally, the com-

```
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] .
```

Fig. 4. Specialization algorithm, Phase 2: Compressed theory $\mathscr{E}''$.

puted renaming is also applied to $\mathscr{R}$ 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 $\mathscr{R}$. This allows the (renamed) rewrite rules to be able to access the new specialized encryption and decryption functionality provided by $\mathscr{E}''$.

Example 1 shows that a high degree of simplification can be achieved by the specialization technique of Presto for theories that do not have the FVP. Furthermore, in many cases, the specialization algorithm is also able to transform an equational theory that does not meet the FVP into a specialized one that does. This typically happens when the function calls to be specialized can only be unfolded a finite number of times. Let us see an example.

Example 17. Consider a slight variant of the protocol theory of Example 16 in which messages consist of one single symbol instead of arbitrarily long sequences of symbols. This variant can be obtained by simply modifying the sort of the messages from `Message` to `Symbol` in the protocol rewrite rules. In this scenario, the set of maximal function calls becomes `Q={enc(S,s(s(0))`$\downarrow_{\vec{E},B}$`,dec(S,s(s(0)`$\downarrow_{\vec{E},B}$`)}`, where `S` is a variable of sort `Symbol`. Note that the calls in `Q` subsume a finite number of more specific calls that correspond to the encryption and decryption of the symbols `a`, `b` and `c` w.r.t. the key `s(s(0))`. The rewrite theory can be automatically specialized by `Presto` for this use case by using $\mathrm{NPER}^{\mathscr{U}}_{\mathscr{A}}(\mathscr{R})$, instantiated with $\mathscr{U} = \mathscr{U}_{\overline{fvp}}$ and $\mathscr{A} = \mathscr{A}_{Elgg}$. The final outcome produced is a specialized rewrite theory $\mathscr{R}'$ whose underlying, transformed equational theory is shown in Figure 5. This theory clearly meets the FVP since it specifies four non-recursive functions that all work over the finite domain `{a,b,c}`. Additionally, the obtained specialization gets rid of the associative data structure needed to build messages of arbitrary size since only one-symbol messages are allowed in the specialized program.

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

Fig. 5. Equations of the specialized equational theory for one-symbol messages and key `K=s(s(0))`.

Finally, note that the satisfaction of the FVP allows narrowing-based reachability problems to be effectively solved within the specialized rewrite theory, while they were unfeasible in the original rewrite theory.[14] For instance, the following reachability goal succeeds, proving that it is possible to establish a successful communication from an initial state in which client `Cli-A` sends a request containing the crypted message `c` to server `Srv-A`

```
< [Cli-A,Srv-A,Q,K,mt] & [Srv-A,K] & (Srv-A <- {Cli-A,c}) > =>*
                                  < [Srv-A,K] & [Cli-A,Srv-A,Q,K,success] >
```

Also the computed substitution `{K/s(s(0)),Q/a}` provides the required key `K` and the plain message `Q` that the server sends back to the client.

(b) When the equational theory $\mathscr{E}$ does satisfy the FVP, FV-narrowing trees are always finite objects that can be effectively constructed in finite time. Therefore, in this specific case, we define the following unfolding operator that constructs the complete FV-narrowing tree for any possible call.

Definition 7. *Given the equational theory $\mathscr{E} = (\Sigma, E \uplus B)$ with a decomposition $\vec{\mathscr{E}} = (\Sigma, B, \vec{E})$, and a set of terms Q to be specialized in $\mathscr{E}$,then*

$$\mathscr{U}_{fvp}(Q, \vec{\mathscr{E}}) = \bigcup_{t \in Q} \{t' \mid t \leadsto^{!}_{\vec{E},B} t' \in VN^{\circlearrowleft}_{\vec{\mathscr{E}}}(t)\}$$

[14] Reachability goals can be solved by the Maude built-in `vu-narrow` command.

where $t \leadsto^{!}_{\vec{E},B} t'$ denotes a FV-narrowing derivation from t to the term t' to which no FV-narrowing steps can be applied.

Note that, when an equational theory has the FVP, both unfolding operators $\mathcal{U}_{fvp}$ and $\mathcal{U}_{\overline{fvp}}$ can be used to specialize a rewrite theory. Nonetheless, the advantage of using $\mathcal{U}_{fvp}$ instead of $\mathcal{U}_{\overline{fvp}}$ is twofold. First, $\mathcal{U}_{fvp}$ disregards any embedding check, which can be extremely time-consuming when $\mathcal{E}$ includes several operators that obey complex modular combinations of algebraic axioms.[15] Second, $\mathcal{U}_{fvp}$ exhaustively explores the whole FV-narrowing tree of a term, while $\mathcal{U}_{\overline{fvp}}$ does not. This leads to a lower degree of specialization when $\mathcal{U}_{\overline{fvp}}$ is applied to a finite variant theory, as shown in the following (pathological) example.

Example 18. Consider the equational theory that is encoded by the following Maude functional module:

```
fmod MKEVEN is sort Nat .
  ops 0 1 : -> Nat [ctor] .  op mkEven : Nat Nat -> Nat .
  op_+_ : Nat Nat -> Nat [ctor assoc comm id: 0] .
  vars X Y : Nat .
  eq mkEven(X + X + 1,1 + Y) = mkEven(X + X + 1 + 1,Y) [variant] .
  eq mkEven(X + X,0) = X + X [variant] .
endfm
```

The equational theory specifies the encoding for natural numbers in Presburger's style[16] and uses this encoding to define the function `mkEven(X,Y)` that makes `X` even (if `X` is odd) by "moving" one unit from `Y` to `X`. Otherwise, if `X` is even, `X` is left unchanged. The partial evaluation of the given theory w.r.t. the call `mkEven(X,Y)` yields different outcomes that depend on the chosen unfolding operator. On the one hand, by using $\mathcal{U}_{\overline{fvp}}$, the output is the very same input theory, thus no "real" specialization is achieved. On the other hand, the unfolding operator $\mathcal{U}_{fvp}$ produces the specialized definition of `mkEven` given by

```
eq mkEven(X + X,0) = X + X [variant] .
eq mkEven(1 + X + X, 1) = 1 + 1 + X + X [variant] .
```

where the second equation is generated by fully exploring the FV-narrowing derivation

$$\mathtt{mkEven(Z,W)} \leadsto_{\{\mathtt{Z/X+X+1,W/1+Y}\}} \mathtt{mkEven(1 + 1 + X + X,Y)} \leadsto_{\{\mathtt{Y/0}\}} \mathtt{1 + 1 + X + X}$$

where $t \leadsto_{\sigma} t$ denotes a FV-narrowing step from t to t' with substitution σ. In contrast, $\mathcal{U}_{\overline{fvp}}$ stops the sequence at `mkEven(1 + 1 + X + X, Y)` since `mkEven(1 + 1 + X + X, Y)` embeds `mkEven(Z,W)` and hence yields a specialized equation that is equal (modulo associativity and commutativity of +) to the original equation `mkEven(X + X + 1,1 + Y) = mkEven(X + X + 1 + 1,Y)`.

Termination of the symbolic specialization Algorithm $\mathrm{NPER}^{\mathcal{U}}_{\mathcal{A}}(\mathcal{R})$ for finite variant theories and non-finite variant theories follows from the global termination of the $\mathrm{EQNPE}^{\mathcal{U}}_{\mathcal{A}}(\mathcal{E})$ algorithm proved in [6] for the unfolding function $\mathcal{U}_{\overline{fvp}}(Q,\vec{\mathcal{E}})$ and the equational least general abstraction function $\mathcal{A}_{Elgg}(Q,\mathcal{L},B)$, and from the local termination of $\mathcal{U}_{fvp}(Q,\vec{\mathcal{E}})$.

[15] Actually, given an AC operator $\circ$, if we want to check whether a term $t = t_1 \circ t_2 \ldots \circ t_i$ is embedded into another term with a similar form, all possible permutations of the elements of both terms must be non-deterministically tried.

[16] Using this encoding, a natural number can be the constant `0` or a sequence of the form `1 + 1 ... + 1`.

Theorem 4 (Termination of $\text{NPER}^{\mathcal{U}}_{\mathcal{A}}(\mathcal{R})$**).** *Let* $\mathcal{R} = (\Sigma, E \uplus B, R)$ *be a rewrite theory such that* $\mathcal{E} = (\Sigma, E \uplus B)$*, and* $\vec{\mathcal{E}} = (\Sigma, B, \vec{E})$ *is a decomposition of* $\mathcal{E}$*. Let* $\mathcal{U}$ *be an unfolding operator and let* $\mathcal{A}$ *be an abstract operator. Algorithm* $\text{NPER}^{\mathcal{U}}_{\mathcal{A}}(\mathcal{R})$ *terminates when* $\mathcal{E}$ *satisfies the FVP (respectively,* $\mathcal{E}$ *does not satisfy the FVP) with the tandem* $\mathcal{U}_{fvp}/\mathcal{A}_{Elgg}$ *(respectively, the tandem* $\mathcal{U}_{\overline{fvp}}/\mathcal{A}_{Elgg}$*).*

6 Experimental Evaluation

Table 1 contains the experiments that we have performed with the Presto system [47], which is a prototype implementation of the proposed specialization framework for rewrite theories. The experimental evaluation has been conducted on an Intel Xeon E5-1660 3.3GHz CPU with 64 GB RAM running Maude v3.1

We have considered the following benchmark programs: *Crypto Protocol*, a variant of the client server communication protocol of Example 1 where we introduce extra functions (e.g., the Fibonacci function and the `mod` function that computes the remainder of the division between natural numbers) in the underlying equational theory to make key generation heavier; *FM-Account*, a rewrite theory that specifies a bank account system with managed accounts which automates a simple investment model; *tkEven*, a simple rewrite theory that tuples a couple of calls to the `mkEven` function of Example 18; *Handshake-KMP*, a handshake network protocol in which a client sends an arbitrary long and noisy message `M` to a server. The handshake succeeds if the server can recognize a secret handshake sequence `P` inside the client message `M` by matching `P` against `M` via the well-known *KMP* string matching algorithm. These experiments plus several others are also publicly available at Presto's website. For each experiment, we publish the source code of all of the examples to make the experiments easily reproducible. In many cases, we transform a rewrite theory whose operators obey algebraic axioms, such as associativity, commutativity, and unity, into a much simpler rewrite theory with no structural axioms.

The experiments have been divided into two classes. The first class, which is identified by the ***Rewriting*** section in Table 1, aims at measuring the speedup of the obtained specialization w.r.t. the rewriting evaluation mechanism; the second class corresponds to the ***Narrowing*** section in Table 1 and measures the speedup w.r.t. narrowing computations. Column $Rls_{\mathcal{R}}/Eqs_{\mathcal{R}}$ (resp. $Rls_{\mathcal{R}'}/Eqs_{\mathcal{R}'}$) shows the size of the original (resp. specialized) theories measured as the number of rules and equations they contain. We do not benchmark the specialization times since they are almost negligible (<100 ms for most cases).

To evaluate rewriting performance, we considered rewrite sequences of increasing sizes that range from 1 million to 10 million rewrite steps. Column *Rewrites* in Table 1 indicates the number of rewrites for each benchmark program and experiment.

For each rewriting experiment, we executed the original specification $\mathcal{R}$ and the specialized $\mathcal{R}'$ on the very same input states. Specialization was performed according to the nature of each benchmark program. Therefore, programs that do not meet the FVP have been specialized using the unfolding operator $\mathcal{U}_{\overline{fvp}}$, while programs that include a finite variant theory have been specialized using the $\mathcal{U}_{fvp}$ operator. We recorded the following experimental data: the execution times (in ms) for $\mathcal{R}$ (Column $T^{\rightarrow}_{\mathcal{R}}$) and for $\mathcal{R}'$ (Column $T^{\rightarrow}_{\mathcal{R}'}$), and the speedup that is computed as the ratio $T^{\rightarrow}_{\mathcal{R}}/T^{\rightarrow}_{\mathcal{R}'}$ (Column $Speedup^{\rightarrow}$). To reduce the noise, we considered the average time of ten runs for each experiment.

Regarding rewriting times, our figures show that the specialized rewrite theories achieve a significant improvement when compared to the original theory, with an average speedup for

	Program Size		***Rewriting***				***Narrowing***			
	$Rls_{\mathscr{R}}/Eqs_{\mathscr{R}}$	$Rls_{\mathscr{R}'}/Eqs_{\mathscr{R}'}$	*Rewrites*	$T^{\rightarrow}_{\mathscr{R}}$ *(ms)*	$T^{\rightarrow}_{\mathscr{R}'}$ *(ms)*	$Speedup^{\rightarrow}$	*Levels*	$T^{\rightsquigarrow}_{\mathscr{R}}$ *(ms)*	$T^{\rightsquigarrow}_{\mathscr{R}'}$ *(ms)*	$Speedup^{\rightsquigarrow}$
Crypto Protocol	3/31	3/6	1M	45,869	953	48.13	50	31,587	3,926	8.05
			2.5M	104,295	2,324	44.88	75	100,124	11,767	8.51
			5M	207,558	4,924	42.15	100	227,459	25,236	9.01
			7.5M	311,016	7,647	40.67	125	426,466	47,388	9.00
			10M	420,795	9,434	44.60	150	730,430	79,738	9.16
Handshake-KMP	2/14	2/0	1M	87,362	576	151.67	2	7	4	1.75
			2.5M	209,143	1,408	148.54	4	49	28	1.75
			5M	416,655	2,875	144.92	6	313	180	1.74
			7.5M	642,196	5,181	123.95	8	1,672	948	1.76
			10M	806,152	6,701	120.30	10	8,008	4,513	1.77
FM-Account	3/11	3/2	1M	14,512	282	51.46	1	1	1	1.00
			2.5M	36,430	649	56.13	2	13	4	3.25
			5M	73,548	1,297	56.71	3	158	16	9.88
			7.5M	108,195	1,977	54.73	4	1,957	49	39.94
			10M	142,475	2,595	54.90	5	35,834	145	247.13
tkEven	1/8	1/8	1M	1,752	383	4.57	5	460	20	23.00
			2.5M	4,276	886	4.83	10	6,129	80	76.61
			5M	8,961	1,907	4.70	15	38,675	183	211.34
			7.5M	13,779	3,044	4.53	20	153,273	337	454.82
			10M	17,612	3,915	4.50	25	492,588	603	816.90

Table 1. Rewriting and narrowing experimental results in Presto.

these benchmarks of 60.34. In other words, the specialized program is, on average, more than 60 times faster than the original program on the considered inputs. Particularly remarkable is the performance improvement of the *Handshake-KMP*, which reaches two orders of magnitude for the case when the specialized maximal calls within the rewrite theory partially instantiate the input pattern and input message. This is basically due to the huge simplification that is achieved by Presto by specializing the *KMP* string matching function. The smallest speedup for rewriting executions is obtained by *tkEven*. In this case, we specialize the very general call `tkEven(X:Nat,Y:Nat)`, which offers less opportunities for optimization since no argument in *tkEven* has been sufficiently instantiated.

For the narrowing experiments, we considered a narrowing-based reachability goal for each benchmark program, which has been used to search for solutions in a narrowing tree fragment of an increasing number of levels from 1 to 150 levels. This means that our experiments consider huge search spaces that consist of complete narrowing trees of depth up to 150 levels. It is worth recalling that narrowing-based reachability goals cannot be solved in those Maude programs with an equational theory $\mathscr{E}$ that does not meet the FVP since $\mathscr{E}$-unification may not terminate. Thus, to be able to execute narrowing-based reachability goals in any benchmark programs $\mathscr{R} = (\Sigma, E \uplus B, R)$ regardless of their FVP behavior, we only performed narrowing steps with R modulo $E' \uplus B$, where $\mathscr{E}' = (\Sigma, E' \uplus B)$ is the maximal FVP-fragment of the equational theory $(\Sigma, E \uplus B)$ included in $\mathscr{R}$. The rest of the equations $(E \setminus E')$ are only used for normalization as described in Section 4.

In this case, we recorded the search time in the corresponding narrowing tree fragment for the original program (Column $T^{\rightsquigarrow}_{\mathscr{R}}$) as well as for the specialized program (Column $T^{\rightsquigarrow}_{\mathscr{R}'}$), together with the achieved speedup (Column $Speedup^{\rightsquigarrow}$). In all narrowing experiments, the specialized program outperforms the original one by greatly reducing the time required to solve the considered narrowing-based reachability goals. The average speedup is 96.82. For some example programs, we note that the speedup for narrowing exponentially grows with the size of the narrowing tree. In particular, this happens for programs *FM-Account* and *tkEven* where the obtained specializations greatly reduce the branching factor of the narrowing trees (both at the level of rules and equations) associated with the considered reachability goals, thereby enabling a faster exploration of the search space in the specialized programs that is more noticeable the larger the size of the trees. This means that some costly analyses that might require inordinate resources, both in time and space, could be effectively performed after the transformation. For the case of *Handshake-KMP*, the performance gain that is obtained for narrowing with specialized rules and equations is much lower than for equational rewriting computations. This is actually expected, since in order to select the maximal FVP-fragment of the *KMP* equational theory (so that we can run the three-level narrowing modulo equations and axioms mechanism), we must remove the `variant` attribute from most of the *KMP* equations, which dramatically reduces the opportunities for optimizing the narrowing computations.

The original EqNPE framework of [6] was implemented in our earlier partial evaluator Victoria. Since Victoria cannot handle rewrite theories but only equational theories, the specialization of rewrite theories supported by Presto could not be previously achieved unless a complex hack is introduced not only at the level of the theory signature but also by providing a suitable program infrastructure that simulates rewrite rule nondeterminism through equations. For the case when we simply specialize an equational theory, both systems perform similarly. For instance, we benchmarked the total specialization time for a loop consisting of 1000 specializations of the *KMP* program that is classically used to compare program specializers, and then we divided the accumulated specialization time by 1000. This modus operandi allowed the noise on the small time for a single specialization to be reduced. The resulting specialization times of Victoria and Presto are comparable (6.6 ms and 5.6 ms, respectively). This is noteworthy since the implementation of Presto is considerably more ambitious and it provides much greater coverage of the Maude language compared to the simpler approach of Victoria.

Presto includes a FVP checker for equational theories that is based on the checking procedure described in [39]. It also implements a strong irreducibility checker that determines whether there is any redex within the left-hand sides of the rewrite rules (w.r.t. the hosted equational theory).

7 Related work and Conclusion

The generic specialization framework proposed in this work represents our most ambitious automated optimization scheme for rewrite theories. To efficiently achieve aggressive specialization that scales to real-world problems, the key components of the EqNPE scheme needed to be thoroughly investigated, extended, and highly optimized over the years. This is because equational problems such as order-sorted equational homeomorphic embedding and order-sorted equational least general anti-unification are much more costly than their corresponding "syntactic" counterparts and achieving proper formalizations and efficient implementations has required years [12, 13, 9, 10, 4, 6, 14].

Our specialization technique can have a decisive impact on the symbolic analysis of concurrent systems that are modeled as rewrite theories in Maude. The main reason why our technique is so effective in this area is that it not only achieves huge speedup for relevant classes of rewrite theories, but it can also cut down an infinite folding variant narrowing space to a finite one for the underlying equational theory $\mathcal{E}$. By doing this, any $\mathcal{E}$-unification problem can be finitely solved, and symbolic, narrowing-based analysis with R modulo $\mathcal{E}$ can be effectively performed.

Among the wide literature on logic program specialization, the partial evaluation of functional logic programs [17, 2, 33] is the closest to our work. The *narrowing-driven Partial Eval-*

uation (NPE) algorithm of [17] extends to narrowing the classical PD scheme of [37] and was proved to be strictly more powerful than the PE of both logic programs and functional programs [17], with a potential for specialization that is comparable to conjunctive partial deduction (CPD) and positive supercompilation [25]. Early instances of this framework implemented partial evaluation algorithms for different narrowing strategies, including lazy narrowing [15], innermost narrowing [17], and needed narrowing [3, 18].

NPE was extended in [6] to the specialization of *order-sorted equational theories* and implemented in the partial evaluator for equational theories Victoria. For a detailed discussion of the literature related to narrowing-driven partial evaluation, we refer to [6].

A narrowing-based partial evaluator for the lazy functional logic language Curry is described in [33, 46]. Its implementation can be seen as an instance of the generic narrowing-based partial evaluation framework of [17]. This system improves the former prototype of [2] by taking into account (mutually recursive) let expressions and non-deterministic operations, while the PE system of [2] was restricted to confluent programs. Obviously, the protocol benchmarks in this paper cannot be directly specialized by using Curry's partial evaluator since neither evaluation modulo algebraic axioms nor concurrency are supported by Curry's partial evaluator; this would require artificially rewriting the program code so that any comparison would be meaningless. In the opposite direction, Presto cannot manage the specialization of higher-order functions that is achieved by [33].

Our specialization technique falls into the category of the semantic-preserving program transformations. There are very few semantic-preserving transformations for rewrite theories in the related literature. An important example is explicit coherence completion [52] between rules, equations and axioms, which is necessary for symbolic execution in rewrite theories and relies on semantically-equivalent theory transformations [41]. Also the semantic $\mathbb{K}$-framework [48] and the model transformations of [49] are based on sophisticated program transformations and both preserve the reduction semantics of the original theory. Nonetheless, they do not aim to program optimization. Furthermore, Maude tools usually rely on weaker theory transformations that preserve only specific properties such as invariants or termination behavior (Full-Maude [22], Real-time Maude [45], MTT [27], and Maude-NPA [28] are prominent examples). Other transformations focus on reducing the size of the search space; for instance, *equational abstraction* [42, 19] reduces an infinite state system to a finite quotient of the original system algebra by introducing some extra equations that preserve certain temporal logic properties.

Finally, we would like to discuss the limitations of the current specialization framework (and its associated implementation) along with possible lines of future work. Although a wide class of Maude rewrite theories can be already partially evaluated, there are still some Maude features that cannot be handled by Presto.

First, there is no support for user-defined rewrite strategies and object-oriented programming. Since these are important Maude features, as future work we plan to investigate possible extensions of the framework that can deal with them.

Second, rewriting logic is parameterized by an equational logic that, in the case of Maude, is membership equational logic (MEL). MEL allows sorts of terms to be asserted through suitable (conditional) membership axioms. Nonetheless, our specialization technique only handles order-sorted equational logic which is strictly less expressive than MEL. This is because the computation of term variants is currently formulated and implemented in Maude only for order-sorted specifications. Hence, the extension of our partial evaluation framework to deal with MEL specifications is far from trivial since it requires to extend both folding variant narrowing and variant computation to MEL specifications.

Third, we plan to generalize our specialization scheme so that it can cope with rewrite theories that are not strongly irreducible. Strong irreducibility is a reasonable requirement that is much more practical and less demanding than a constructor discipline forbidding that arguments of the left-hand side of rules contain any defined symbols. Nonetheless, we do believe that more relaxed conditions can be found to specialize rewrite theories without jeopardizing correctness of the specialization.

Bibliography

[1] E. Albert, M. Alpuente, M. Falaschi, P. Julián, and G. Vidal. Improving Control in Functional Logic Program Specialization. In *Proceedings of the 5th International Symposium on Static Analysis (SAS 1998)*, volume 1503 of *Lecture Notes in Computer Science*, pages 262–277. Springer, 1998.

[2] E. Albert, M. Hanus, and G. Vidal. A Practical Partial Evaluation Scheme for Multi-Paradigm Declarative Languages. *Journal of Functional and Logic Programming*, 2002:1–34, 2002.

[3] Elvira Albert, María Alpuente, Michael Hanus, and Germán Vidal. A Partial Evaluation Framework for Curry Programs. In Harald Ganzinger, David A. McAllester, and Andrei Voronkov, editors, *Logic Programming and Automated Reasoning, 6th International Conference, LPAR'99, Tbilisi, Georgia, September 6-10, 1999, Proceedings*, volume 1705 of *Lecture Notes in Computer Science*, pages 376–395. Springer, 1999.

[4] M. Alpuente, D. Ballis, A. Cuenca-Ortega, S. Escobar, and J. Meseguer. ACUOS^2: A High-Performance System for Modular ACU Generalization with Subtyping and Inheritance. In *Proceedings of the 16th European Conference on Logics in Artificial Intelligence (JELIA 2019)*, volume 11468 of *Lecture Notes in Computer Science*, pages 171–181. Springer, 2019.

[5] M. Alpuente, D. Ballis, S. Escobar, and J. Sapiña. Narrowing-based Optimization of Rewrite Theories. In *Proceedings of the 7th International Workshop on Rewriting Techniques for Program Transformations and Evaluation (WPTE 2020)*, 2020. Extended version in *Tech. Report DSIC-UPV, 2020.* Available at: `http://elp.webs.upv.es/papers/ABES20-TR.pdf`.

[6] M. Alpuente, A. Cuenca-Ortega, S. Escobar, and J. Meseguer. A Partial Evaluation Framework for Order-Sorted Equational Programs modulo Axioms. *Journal of Logical and Algebraic Methods in Programming*, 110:1–36, 2020.

[7] M. Alpuente, A. Cuenca-Ortega, S. Escobar, and J. Meseguer. Order-sorted Homeomorphic Embedding modulo Combinations of Associativity and/or Commutativity Axioms. *Fundamenta Informaticae*, 177(3-4):297–329, 2020.

[8] M. Alpuente, A. Cuenca-Ortega, S. Escobar, and J. Sapiña. Inspecting Maude Variants with GLINTS. *Theory and Practice of Logic Programming*, 17(5–6):689–707, 2017.

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

[10] M. Alpuente, S. Escobar, J. Espert, and J. Meseguer. ACUOS: A System for Modular ACU Generalization with Subtyping and Inheritance. In *Proceedings of the 14th European Conference on Logics in Artificial Intelligence (JELIA 2014)*, volume 8761 of *Lecture Notes in Computer Science*, pages 573–581. Springer, 2014.

[11] M. Alpuente, S. Escobar, and J. Iborra. Termination of Narrowing Revisited. *Theoretical Computer Science*, 410(46):4608–4625, 2009.

[12] M. Alpuente, S. Escobar, J. Meseguer, and P. Ojeda. A Modular Equational Generalization Algorithm. In *Proceedings of the 18th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR 2008)*, volume 5438 of *Lecture Notes in Computer Science*, pages 24–39. Springer, 2008.

[13] M. Alpuente, S. Escobar, J. Meseguer, and P. Ojeda. Order-Sorted Generalization. *Electronic Notes in Theoretical Computer Science*, 246:27–38, 2009.

[14] M. Alpuente, S. Escobar, J. Meseguer, and J. Sapiña. Order-sorted Equational Generalization Algorithm Revisited, 2021. *Submitted for publication.* Available at: `elp.webs.upv.es/papers/elgg-rev.pdf`.

[15] M. Alpuente, M. Falaschi, P. Julián, and G. Vidal. Specialization of Lazy Functional Logic Programs. In *Proceedings of the ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation (PEPM 1997)*, pages 151–162. Association for Computing Machinery, 1997.

[16] M. Alpuente, M. Falaschi, G. Moreno, and G. Vidal. Safe Folding/Unfolding with Conditional Narrowing. In *Proceedings of the 6th International Joint Conference on Algebraic and Logic Programming (ALP 1997)*, volume 1298 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 1997.

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

[18] M. Alpuente, S. Lucas, M. Hanus, and G. Vidal. Specialization of Functional Logic Programs based on Needed Narrowing. *Theory and Practice of Logic Programming*, 5(3):273–303, 2005.

[19] K. Bae, S. Escobar, and J. Meseguer. Abstract Logical Model Checking of Infinite-State Systems Using Narrowing. In *Proceedings of the 24th International Conference on Rewriting Techniques and Applications (RTA 2013)*, volume 21 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 81–96. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013.

[20] C. Bouchard, K. A. Gero, C. Lynch, and P. Narendran. On Forward Closure and the Finite Variant Property. In *Proceedings of the 9th International Symposium on Frontiers of Combining Systems (FroCos 2013)*, volume 8152 of *Lecture Notes in Computer Science*, pages 327–342. Springer, 2013.

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

[22] M. Clavel, F. Durán, S. Eker, S. Escobar, P. Lincoln, N. Martí-Oliet, J. Meseguer, R. Rubio, and C. Talcott. Maude Manual (Version 3.0). Technical report, SRI International Computer Science Laboratory, 2020. Available at: `http://maude.cs.uiuc.edu`.

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

[24] H. Comon-Lundh and S. Delaune. The Finite Variant Property: How to Get Rid of Some Algebraic Properties. In *Proceedings of the 16th International Conference on Rewriting Techniques and Applications (RTA 2005)*, volume 3467 of *Lecture Notes in Computer Science*, pages 294–307. Springer, 2005.

[25] D. De Schreye, R.Glück, J. Jørgensen, M. Leuschel, B. Martens, and M. H. Sørensen. Conjunctive Partial Deduction: Foundations, Control, Algorithms, and Experiments. *The Journal of Logic Programming*, 41(2-3):231–277, 1999.

[26] F. Durán, S. Eker, S. Escobar, N. Martí-Oliet, J. Meseguer, and C. Talcott. Associative Unification and Symbolic Reasoning Modulo Associativity in Maude. In *Proceedings of the 12th International Workshop on Rewriting Logic and its Applications (WRLA 2018)*, volume 11152 of *Lecture Notes in Computer Science*, pages 98–114. Springer, 2018.

[27] F. Durán, S. Lucas, and J. Meseguer. MTT: The Maude Termination Tool (System Description). In *Proceedings of the 4th International Joint Conference on Automated Rea-*

soning (IJCAR 2008), volume 5195 of *Lecture Notes in Computer Science*, pages 313–319. Springer, 2008.

[28] S. Escobar, C. Meadows, and J. Meseguer. Maude-NPA: Cryptographic Protocol Analysis Modulo Equational Properties. In *Foundations of Security Analysis and Design V (FOSAD 2007/2008/2009 Tutorial Lectures)*, volume 5705 of *Lecture Notes in Computer Science*, pages 1–50. Springer, 2009.

[29] S. Escobar and J. Meseguer. Symbolic Model Checking of Infinite-State Systems Using Narrowing. In *Proceedings of the 18th International Conference on Term Rewriting and Applications (RTA 2007)*, volume 4533 of *Lecture Notes in Computer Science*, pages 153–168. Springer, 2007.

[30] S. Escobar, J. Meseguer, and R. Sasse. Variant Narrowing and Equational Unification. *Electronic Notes in Theoretical Computer Science*, 238(3):103–119, 2009.

[31] S. Escobar, R. Sasse, and J. Meseguer. Folding Variant Narrowing and Optimal Variant Termination. *The Journal of Logic and Algebraic Programming*, 81(7–8):898–928, 2012.

[32] M. Fay. First Order Unification in an Equational Theory. In *Proceedings of the 4th International Conference on Automated Deduction (CADE 1979)*, pages 161–167. Academic Press, Inc., 1979.

[33] M. Hanus and B. Peemöller. A Partial Evaluator for Curry. In *Proceedings of the 28th International Workshop on (Constraint) Logic Programming (WLP 2014)*, volume 1335, pages 155–171. CEUR-WS.org, 2014.

[34] N. D. Jones, C. K. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice-Hall, 1993.

[35] M. Leuschel. Improving Homeomorphic Embedding for Online Termination. In *Proceedings of the 8th International Workshop on Logic Programming Synthesis and Transformation (LOPSTR 1998)*, volume 1559 of *Lecture Notes in Computer Science*, pages 199–218. Springer, 1998.

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

[37] B. Martens and J. Gallagher. Ensuring Global Termination of Partial Deduction while Allowing Flexible Polyvariance. In *Proceedings of the 12th International Conference on Logic Programming (ICLP 1995)*, pages 597–611. The MIT Press, 1995.

[38] J. Meseguer. Conditional Rewriting Logic as a Unified Model of Concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.

[39] J. Meseguer. Variant-Based Satisfiability in Initial Algebras. In *Proceedings of the 4th International Workshop for Safety-Critical Systems (FTSCS 2015)*, volume 596 of *Communications in Computer and Information Science*, pages 3–34. Springer, 2015.

[40] J. Meseguer. Strict Coherence of Conditional Rewriting Modulo Axioms. *Theoretical Computer Science*, 672:1–35, 2017.

[41] J. Meseguer. Generalized Rewrite Theories, Coherence Completion, and Symbolic Methods. *Journal of Logical and Algebraic Methods in Programming*, 110, 2020.

[42] J. Meseguer, M. Palomino, and N. Martí-Oliet. Equational Abstractions. *Theoretical Computer Science*, 403(2–3):239–264, 2008.

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

[44] A. Middeldorp and E. Hamoen. Counterexamples to Completeness Results for Basic Narrowing. In *Proceedings of the 3rd International Conference on Algebraic and Logic Pro-*

gramming (ALP 1992), volume 632 of *Lecture Notes in Computer Science*, pages 244–258. Springer, 1992.

[45] P. C. Ölveczky and J. Meseguer. The Real-Time Maude Tool. In *Proceedings of the 14th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS 2008)*, volume 4963 of *Lecture Notes in Computer Science*, pages 332–336. Springer, 2008.

[46] B. Peemöller. *Normalization and Partial Evaluation of Functional Logic Programs*. PhD thesis, University of Kiel, Germany, 2017.

[47] The Presto Website, 2020. Available at: `http://safe-tools.dsic.upv.es/presto`.

[48] G. Roşu. $\mathbb{K}$: A Semantic Framework for Programming Languages and Formal Analysis Tools. In *Dependable Software Systems Engineering*, volume 50 of *NATO Science for Peace and Security Series - D: Information and Communication Security*, pages 186–206. IOS Press, 2017.

[49] A. Rodríguez, F. Durán, A. Rutle, and L. M. Kristensen. Executing Multilevel Domain-Specific Models in Maude. *Journal of Object Technology*, 18(2):4:1–21, 2019.

[50] S. Skeirik and J. Meseguer. Metalevel Algorithms for Variant Satisfiability. *Journal of Logical and Algebraic Methods in Programming*, 96:81–110, 2018.

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

[52] P. Viry. Equational Rules for Rewriting Logic. *Theoretical Computer Science*, 285(2):487–517, 2002.

A Proofs of Technical Results

In this appendix, we demonstrate the main technical results of the paper, together with other important results for equational partial evaluation.

In the following, given a substitution σ and a renaming ρ, we define $RN_\rho(\sigma)(x) = RN_\rho(\sigma(x))$ for $x \in Dom(\sigma)$ (renaming of a substitution). We recall the following two main results of [6].

Theorem 5 (Preservation of executability conditions by $\text{EQNPE}^{\mathscr{U}}_{\mathscr{A}}(\mathscr{E},Q)$ **[6]).** *Let $\mathscr{E} = (\Sigma, E \uplus B)$ be an equational theory such that $\vec{\mathscr{E}} = (\Sigma, B, \vec{E})$ is a decomposition of $\mathscr{E}$, let u be a Σ-term, and let Q be a finite set of Σ-terms.*

Given an unfolding operator $\mathscr{U}$ and an abstract operator $\mathscr{A}$, let $Q' = \text{EQNPE}^{\mathscr{U}}_{\mathscr{A}}(\mathscr{E},Q)$ and let $\mathscr{E}' = (\Sigma', E' \uplus B') = \text{GENTHEORY}(Q',(\Sigma, E \uplus B))$ be the partial evaluation of $\mathscr{E}$ w.r.t Q. Then, $\vec{\mathscr{E}}'$ is a decomposition of $\mathscr{E}'$.

Theorem 6 (Strong Correctness and Completeness of $\text{EQNPE}^{\mathscr{U}}_{\mathscr{A}}(\mathscr{E},Q)$ **[6]).** *Let $\mathscr{E} = (\Sigma, E \uplus B)$ be an equational theory such that $\vec{\mathscr{E}} = (\Sigma, B, \vec{E})$ is a decomposition of $\mathscr{E}$, and let Q be a finite set of Σ-terms.*

Given an unfolding operator $\mathscr{U}$ and an abstract operator $\mathscr{A}$, let $Q' = \text{EQNPE}^{\mathscr{U}}_{\mathscr{A}}(\mathscr{E},Q)$ and let $\mathscr{E}' = (\Sigma', E' \uplus B') = \text{GENTHEORY}(Q',(\Sigma, E \uplus B))$ is the partial evaluation of $\mathscr{E}$ w.r.t Q. Let u be a Σ-term, let ρ be an independent renaming of Q', and let $u' = RN_\rho(u)$. If u is B-closed w.r.t. Q' and Σ, then $(u \leadsto^*_{\sigma,\vec{E},B} v) \in VN^{\circlearrowleft}_{\vec{\mathscr{E}}}(u)$ for a variant v that is B-closed w.r.t. Q' and Σ, if and only if $(u' \leadsto^*_{\sigma',\vec{E}',B} v') \in VN^{\circlearrowleft}_{\vec{\mathscr{E}}'}(u')$, where $v' =_B RN_\rho(v)$ and $\sigma' \leq_B RN_\rho(\sigma)$.

Theorem 3. *The operator $\mathscr{A}_{Elgg}(Q,T,B)$ terminates and is an abstraction operator in the sense of Definition 3.*

Proof. **(Proof sketch)** The proof is a slight modification of the proof of Proposition 18 of [6] that considers subsort overloading. Note that Condition (1) of Definition 3 is trivially fulfilled, since $abs^{\check{\trianglelefteq}_B}$ only applies the lgg_B operator, which cannot introduce function symbols not appearing in Q or T. Condition (2) is proved by well-founded, structural induction on $\mathscr{M}_{Q\cup T}$, which is the multiset of all the depths of the terms in $Q \cup T$. The induction entirely resembles the proof in [6] with a minor change in the treatment of the constructor case $abs^{\check{\trianglelefteq}_B}(Q, c(\overline{t_n}))$, where c is a constructor symbol. In the original proof, there is no need to check the least sort of $c(\overline{t_n : s_n})$ since no overloaded operator c could have both a constructor and a defined typing. Here, we relaxed this condition in favor of the milder preregularity-below property and thus the constructor case $abs^{\check{\trianglelefteq}_B}(Q, c(\overline{t_n : s_n}))$ is applied if and only if the following refined check is fulfilled: $c : \overline{s_n} \to s \in \Sigma$ $s.t.$ $c \in \Omega, ls(t) = s$.

Similarly, termination of the abstract operator computation can be proved as in the proof of Theorem 2 of [6]. The proof scheme again exploits a well-founded, structural induction on $\mathscr{M}_{Q\cup T}$.

First, we ensure that executability conditions are preserved for rewrite theories that are specialized by using the $\text{NPER}^{\mathscr{U}}_{\mathscr{A}}$ algorithm.

Lemma 1 (Preservation of decomposition). *Let $\mathscr{R} = (\Sigma, E \uplus B, R)$ be a topmost rewrite theory where $\vec{\mathscr{E}} = (\Sigma, B, \vec{E})$ is a decomposition of $\mathscr{E} = (\Sigma, E \uplus B)$. Given an unfolding operator $\mathscr{U}$ and an abstract operator $\mathscr{A}$, let $\mathscr{R}' = (\Sigma', E' \uplus B', R') = \text{NPER}^{\mathscr{U}}_{\mathscr{A}}(\mathscr{R})$ be a specialization of $\mathscr{R}$ under the independent renaming ρ. Then, $\vec{\mathscr{E}}' = (\Sigma', B', \vec{E}')$ is a decomposition of $\mathscr{E}' = (\Sigma', E' \uplus B')$.*

Proof. By Theorem 5. □

Lemma 2 (Preservation of topmost condition). *Let $\mathscr{R} = (\Sigma, E \uplus B, R)$ be a topmost rewrite theory where $\vec{\mathscr{E}} = (\Sigma, B, \vec{E})$ is a decomposition of $\mathscr{E} = (\Sigma, E \uplus B)$. Given an unfolding operator $\mathscr{U}$ and an abstract operator $\mathscr{A}$, let $\mathscr{R}' = (\Sigma', E' \uplus B', R') = \mathrm{NPER}^{\mathscr{U}}_{\mathscr{A}}(\mathscr{R})$ be a specialization of $\mathscr{R}$ under the independent renaming ρ. Then, $\mathscr{R}'$ is topmost.*

Proof. Immediate, since the topmost property ensures that no function symbol of the signature Σ admits a subterm of the topmost sort *State*. Then, the property follows straightforwardly because no extra sorts are introduced by the independent renaming ρ: given $\rho(t) = f_t(\overline{x_n : s_n})$, where t has sort s, we have $f_t : \overline{s_n} \to s$. □

Lemma 3 (Preservation of coherence). *Let $\mathscr{R} = (\Sigma, E \uplus B, R)$ be a topmost rewrite theory where $\mathscr{E} = (\Sigma, E \uplus B)$, $\vec{\mathscr{E}} = (\Sigma, B, \vec{E})$ is a decomposition of $\mathscr{E}$, R is $\mathscr{E}$-coherent, and the left-hand sides of the rules in R are $(\vec{E}, B)$-strongly irreducible.*

Given an unfolding operator $\mathscr{U}$ and an abstract operator $\mathscr{A}$, let $\mathscr{R}' = (\Sigma', E' \uplus B', R') = \mathrm{NPER}^{\mathscr{U}}_{\mathscr{A}}(\mathscr{R})$ be a specialization of $\mathscr{R}$ under the independent renaming ρ. Then, R' is $\mathscr{E}'$-coherent with $\mathscr{E}' = (\Sigma', E' \uplus B')$.

Proof. Immediate by the condition of $(\vec{E}, B)$-*strong irreducibility* since no equation can be applied to the left-hand side of a rule of R. □

It is very important that equations do not interfere with the topmost property, as shown in the following example.

Example 19. The following rewrite theory does not satisfy the strong irreducibility condition of Lemma 3 because the left-hand side `f(X,Y)` of the rewrite rule can be narrowed by using the equation.

```
mod EXA-COHERENCE is
  sorts AB F .
  ops a b : -> AB [ ctor ] .
  op f : AB AB -> F .
  vars X Y : AB .
  eq f(X,a) = f(X,b) [ variant ] .
  rl f(X,Y) => f(a,Y) [ narrowing ] .
endm
```

The specialization of this rewrite theory proceeds by partially evaluating the two expressions occurring in the rule: `f(X,Y)` and `f(a,Y)`. The partially evaluated equational theory contains one extra variant equation.

```
  eq f(a, a) = f(a, b) [ variant ] .
```

Given the independent renaming $\{$`f(X,b)` $\mapsto$ `f0(X)`, `f(X,Y)` $\mapsto$ `f1(X,Y)`, `f(a,Y)` $\mapsto$ `f2(Y)`, `f(a,b)` $\mapsto$ `f3`$\}$, the specialized rewrite theory is

```
mod EXA-COHERENCE is
  sorts AB F .
  ops a b : -> AB [ ctor ] .
  vars X Y : AB .
  op f0 : AB -> F [ ctor ] .
  op f1 : AB AB -> F .
  op f2 : AB -> F .
  op f3 : -> F [ ctor ] .
  eq f1(X, a) = f0(X) [ variant ] .
  eq f2(a) = f3 [ variant ] .
  rl f1(X, Y) => f2(Y) [ narrowing ] .
endm
```

However, this theory is no longer coherent, and moreover the term `f(b,b)` is reducible in the original rewrite theory but the corresponding renamed term `f0(b)` cannot be reduced in the specialized rewrite theory.

Now we are ready to prove that the specialization of a rewrite theory preserves the executability conditions of the original rewrite theory.

Theorem 1. (Preservation of executability conditions by $\mathrm{NPER}^{\mathscr{U}}_{\mathscr{A}}(\mathscr{R})$**).** *Let $\mathscr{R} = (\Sigma, E \uplus B, R)$ be a topmost rewrite theory such that $\mathscr{E} = (\Sigma, E \uplus B)$ and R is $\mathscr{E}$-coherent. Let $\vec{\mathscr{E}} = (\Sigma, B, \vec{E})$ be a decomposition of $\mathscr{E}$, and let the left-hand sides of the rules in R be $(\vec{E}, B)$-*strongly irreducible*. Let $\mathscr{U}$ be an unfolding operator and let $\mathscr{A}$ be an abstract operator. Given the set Q of the maximal calls in the normalized rules of R, let $\mathscr{R}' = (\Sigma', E' \uplus B', R') = \mathrm{NPER}^{\mathscr{U}}_{\mathscr{A}}(\mathscr{R})$ be the specialization of $\mathscr{R}$, with $Q' = \mathrm{E}\mathrm{Q}\mathrm{NPE}^{\mathscr{U}}_{\mathscr{A}}((\Sigma, E \uplus B), Q)$ and $\mathscr{E}' = (\Sigma', E' \uplus B') = \textsc{GenTheory}(Q', (\Sigma, E \uplus B))$ being the partial evaluation of $\mathscr{E}$ w.r.t Q. (under a given independent renaming ρ for Q').*

*Then, $\vec{\mathscr{E}}' = (\Sigma', B', \vec{E}')$ is a decomposition of $\mathscr{E}'$, R' is $\mathscr{E}'$-coherent, and the left-hand sides of the rules in R' are $(\vec{E}', B')$-*strongly irreducible.

Proof. The result follows immediately from Lemmata 1, 2 and 3 □

In order to prove our main result, we first prove the following auxiliary lemma.

Lemma 4 (Preservation of unification). *Let $\mathscr{R} = (\Sigma, E \uplus B, R)$ be a topmost rewrite theory such that $\mathscr{E} = (\Sigma, E \uplus B)$ and R is $\mathscr{E}$-coherent. Let $\vec{\mathscr{E}} = (\Sigma, B, \vec{E})$ be a decomposition of $\mathscr{E}$, and let the left-hand sides of the rules in R be $(\vec{E}, B)$-*strongly irreducible*. Let $\mathscr{U}$ be an unfolding operator and let $\mathscr{A}$ be an abstract operator. Given the set Q of the maximal calls in the normalized rules of R, let $\mathscr{R}' = (\Sigma', E' \uplus B', R') = \mathrm{NPER}^{\mathscr{U}}_{\mathscr{A}}(\mathscr{R})$ be the specialization of $\mathscr{R}$, with $Q' = \mathrm{E}\mathrm{Q}\mathrm{NPE}^{\mathscr{U}}_{\mathscr{A}}((\Sigma, E \uplus B), Q)$ and $\mathscr{E}' = (\Sigma', E' \uplus B') = \textsc{GenTheory}(Q', (\Sigma, E \uplus B))$ being the partial evaluation of $\mathscr{E}$ w.r.t Q. (under a given independent renaming ρ for Q').*

Let t be a term that is B-closed w.r.t. Q' and Σ and let l be the left-hand side of a rule in R. Then, σ is an $\mathscr{E}$-unifier of t and l if and only if $RN_\rho(\sigma)$ is an $\mathscr{E}'$-unifier of $RN_\rho(t)$ and $RN_\rho(l)$.

Proof. Since l is $(\vec{E}, B)$-*strongly irreducible*, we have that $(t\sigma)\downarrow_{\vec{E},B} =_B l\sigma$. By Theorem 6, l being $(\vec{E}, B)$-strongly irreducible implies $RN_\rho(l)$ is $(\vec{E}', B')$-strongly irreducible. Since t is B-closed w.r.t. Q' and Σ, by Theorem 6, we have that there exists a substitution θ such that $(t \leadsto^*_{\theta,\vec{E},B} l\sigma) \in VN^{\circlearrowleft}_{\vec{\mathscr{E}}}(t)$ if and only if $(t' \leadsto^*_{\theta',\vec{E}',B'} l\sigma') \in VN^{\circlearrowleft}_{\vec{\mathscr{E}}'}(t')$, where $t' = RN_\rho(t)$, $\theta' = RN_\rho(\theta)$, and $\sigma' = RN_\rho(\sigma)$. Therefore, the conclusion follows. □

The following result establishes the strong correctness of the $\mathrm{NPER}^{\mathscr{U}}_{\mathscr{A}}$ specialization Algorithm 1, which states that the specialized rewrite theory $\mathscr{R}' = \mathrm{NPER}^{\mathscr{U}}_{\mathscr{A}}(\mathscr{R})$ and the original theory $\mathscr{R}$ are equivalent in the very strong sense that all computations in $\mathscr{R}$ are preserved in $\mathscr{R}'$.

Theorem 2. (Strong correctness of $\mathrm{NPER}^{\mathscr{U}}_{\mathscr{A}}(\mathscr{R})$**).** *Let $\mathscr{R} = (\Sigma, E \uplus B, R)$ be a topmost rewrite theory such that $\mathscr{E} = (\Sigma, E \uplus B)$ and R is $\mathscr{E}$-coherent. Let $\vec{\mathscr{E}} = (\Sigma, B, \vec{E})$ be a decomposition* *of $\mathscr{E}$, and let the left-hand sides of the rules in R be* $(\vec{E},B)$-strongly irreducible. *Let $\mathscr{U}$ be an unfolding operator and let $\mathscr{A}$ be an abstract operator. Given the set Q of the maximal calls in the normalized rules of R, let $\mathscr{R}' = (\Sigma',E' \uplus B',R') =$ NPER$^{\mathscr{U}}_{\mathscr{A}}(\mathscr{R})$ be the specialization of $\mathscr{R}$, with $Q' =$* EQNPE$^{\mathscr{U}}_{\mathscr{A}}(\mathscr{E},Q)$ *and $\mathscr{E}' = (\Sigma',E' \uplus B') =$* GENTHEORY$(Q',(\Sigma,E \uplus B))$ *being the partial evaluation of $\mathscr{E}$ w.r.t Q. (under a given independent renaming ρ for Q').*

Let $u \in \mathscr{T}_\Sigma(\mathscr{X})$ be B-closed w.r.t. Q' and Σ and $u' = RN_\rho(u) \in \mathscr{T}_{\Sigma'}(\mathscr{X})$.

1. *$(u \rightarrow^*_{R,\vec{E}\uplus B} v)$ if and only if $(u' \rightarrow^*_{R',\vec{E}'\uplus B'} v')$, with $v' =_{B'} RN_\rho(v)$.*
2. *If $\mathscr{E}$ satisfies the FVP, then for any $(\vec{E},B)$-irreducible computed substitution σ, $(u \rightsquigarrow^*_{\sigma,R,\vec{E}\uplus B} v)$ if and only if $(u' \rightsquigarrow^*_{\sigma',R',\vec{E}'\uplus B'} v')$, with $v' =_{B'} RN_\rho(v)$ and $\sigma' =_{B'} RN_\rho(\sigma)$.*

Proof. Immediate by Lemma 4 which works for: (i) matching with only axioms B, and (ii) unification with equations E and axioms B. □

Global termination of the EQNPE$^{\mathscr{U}}_{\mathscr{A}}(\mathscr{E})$ algorithm was proved in [6] for a version of $\mathscr{U}_{\overline{fvp}}(Q,\vec{\mathscr{E}})$ and $\mathscr{A}_{Elgg}(Q,\mathscr{L},B))$ that do not deal with subsort overloading for constructor operators. By replacing these notions with the extended definitions of equational closedness and equational generalization given in this article, the proof of global termination extends with no changes to theories that fully cope with subsort overloading and that are either finite variant or non-finite variant.

More precisely, for the equational theories that do not satisfy the FVP, the equational homeomorphic embedding integrated in $\mathscr{U}_{\overline{fvp}}(Q,\vec{\mathscr{E}})$, which is used to prevent infinite narrowing derivations, is not affected by the extension, and termination of $\mathscr{A}_{Elgg}(Q,\mathscr{L},B)$ is not affected by the extension either. For equational theories that have the FVP, $\mathscr{U}_{fvp}(Q,\vec{\mathscr{E}})$ terminates by definition, and hence the tandem $\mathscr{U}_{fvp}/\mathscr{A}_{Elgg}$ cannot introduce any nontermination issues.

Theorem 7 (Termination of EQNPE$^{\mathscr{U}}_{\mathscr{A}}(\mathscr{E},Q)$**).** *Let $\mathscr{E} = (\Sigma,E \uplus B)$ be an equational theory, and $\vec{\mathscr{E}} = (\Sigma,B,\vec{E})$ be a decomposition of $\mathscr{E}$. Let Q be a set of Σ-terms. Let $\mathscr{U}$ be an unfolding operator and let $\mathscr{A}$ be an abstract operator. Then, Algorithm* EQNPE$^{\mathscr{U}}_{\mathscr{A}}(\mathscr{E},Q)$ *terminates both for the tandem $\mathscr{U}_{fvp}/\mathscr{A}_{Elgg}$ and the tandem $\mathscr{U}_{\overline{fvp}}/\mathscr{A}_{Elgg}$.*

Proof. The result follows from: 1) the fact that $\mathscr{U}_{fvp}(Q,\vec{\mathscr{E}})$ terminates for FVP theories; 2) the local termination of EQNPE for $\mathscr{U}_{\overline{fvp}}(Q,\vec{\mathscr{E}})$ (Theorem 1 in [6]); 3) termination of the abstraction operator $\mathscr{A}_{Elgg}(Q,\mathscr{L},B))$ (Theorem 2 in [6]); and 4) the fact that the proof of the global termination of EQNPE in (Theorem 3 in [6]) relies on: a) termination of the applied unfolding operator and termination of the abstraction operator $\mathscr{A}_{Elgg}(Q,\mathscr{L},B))$. □

Now we can prove the termination result for the symbolic specialization of rewrite theories that are defined on top of both, finite variant and non-finite variant equational theories.

Theorem 4. (Termination of NPER$^{\mathscr{U}}_{\mathscr{A}}(\mathscr{R})$**).** *Let $\mathscr{R} = (\Sigma,E \uplus B,R)$ be a rewrite theory such that $\mathscr{E} = (\Sigma,E \uplus B)$, and $\vec{\mathscr{E}} = (\Sigma,B,\vec{E})$ is a decomposition of $\mathscr{E}$. Let $\mathscr{U}$ be an unfolding operator and let $\mathscr{A}$ be an abstract operator. Algorithm* NPER$^{\mathscr{U}}_{\mathscr{A}}(\mathscr{R})$ *terminates when $\mathscr{E}$ satisfies the FVP (respectively, $\mathscr{E}$ does not satisfy the FVP) with the tandem $\mathscr{U}_{fvp}/\mathscr{A}_{Elgg}$ (respectively, the tandem $\mathscr{U}_{\overline{fvp}}/\mathscr{A}_{Elgg}$).*

Proof. Termination of the Phase 1 follows straightforwardly from Theorem 7. Termination of the Phase 2 is immediate since function RN_ρ trivially terminates (structural induction over terms). □

B Client-Server Communication protocol

This section includes the complete Maude specification of the client-server communication protocol of Example 1.

```
fmod CAESAR-CIPHER is
    pr TRUTH-VALUE .

    sorts Nat NzNat Symbol Message .
    subsort NzNat < Nat .
    subsort Symbol < Message .

    --- Nat constructors and defined functions for natural numbers

    op 0 : -> Nat [ctor] .
    op s : Nat -> NzNat  [ctor] .
    op _<_ : Nat Nat -> Bool .
    op _+_ : Nat Nat -> Nat [assoc comm] .

    --- if then else
    op [_,_,_] : Bool Nat Nat -> Nat .

        vars X Y : Nat .

    --- Addition, lower-than, and if-then-else: definition
    eq 0 + X = X [variant] .
    eq s(X) + Y = s(X + Y) [variant] .
    eq 0 < s(X) = true [variant] .
    eq X < 0 = false [variant] .
    eq s(X) < s(Y) = X < Y [variant] .
    eq [ true,X,Y ] = X [variant] .
    eq [ false,X,Y ] = Y [variant] .

    ---  Alphabet symbols
    op a : -> Symbol [ctor] .
    op b : -> Symbol [ctor] .
    op c : -> Symbol [ctor] .

    --- list constructor (messages)
    op __ : Message Message -> Message [ctor assoc] .


    --- Symbol-to-Nat Nat-to-Symbol operators: declaration and definition
    op toNat : Symbol -> Nat .
    op toSym : Nat -> Symbol .
    op len : ->  Nat .

    eq len = s(s(s(0))) . --- Alphabet cardinality
    eq toNat(a) = 0 [variant] .
    eq toNat(b) = toNat(a) + s(0) [variant] .
    eq toNat(c) = toNat(b) + s(0) [variant] .
    eq toSym(0)= a [variant] .
    eq toSym(s(0)) = b [variant] .
    eq toSym(s(s(0))) = c [variant] .


    --- Encryption/Decryption operators: declaration and definition
    op shift : Nat -> Nat .
    op unshift : Nat -> Nat .
    op en : Nat Nat -> Nat .
    op de : Nat Nat -> Nat .
    op enc : Message Nat -> Message .
    op dec : Message Nat -> Message .

    var M : Message .
    var K : Nat .

    eq shift(X) = [ s(X) < len,s(X), 0 ] [variant] .
    eq unshift(0) = s(s(0)) [variant] .
    eq unshift(s(X)) = X [variant] .

    eq en(X,0) = X [variant] .
    eq en(X,s(Y)) = en(shift(X),Y) [variant] .
    eq de(X,0) = X [variant] .
    eq de(X,s(Y)) = de(unshift(X),Y) [variant] .

    eq enc(S:Symbol,K) = toSym(en(toNat(S:Symbol),K)) [variant] .
    eq enc(S:Symbol M,K) = toSym(en(toNat(S:Symbol),K)) enc(M,K) [variant] .
    eq dec(S:Symbol,K) = toSym(de(toNat(S:Symbol),K))[variant] .
    eq dec(S:Symbol M,K) = toSym(de(toNat(S:Symbol),K)) dec(M,K) [variant] .
endfm
```

```
mod CLI=SERV-PROTOCOL-CAESAR is
    pr CAESAR-CIPHER .
    sorts Content State Packet Cli Serv Host CliName ServName Conf Status.
    subsorts Packet Cli Serv < State .
    subsorts CliName ServName < Host .

    op Srv-A Srv-B : -> ServName [ctor] .
    op Cli-A Cli-B : -> CliName [ctor] .
    op null : -> State [ctor] .
    op _&_ : State State -> State [ctor assoc comm id: null] .
    op _<-_ : Host Content -> Packet [ctor] .
    op {_,_} : Host Message -> Content [ctor] .
    op [_,_,_,_,_] : CliName ServName Message Nat Status -> Cli [ctor] .
    op [_,_] : ServName Nat -> Serv [ctor] .
    op success : -> Status [ctor] .
    op mt : -> Status [ctor] .
    op <_> : State -> Conf [ctor] .

    var K : Nat .
    var C : CliName .
    var S : ServName .
    vars Q M : Message .
    var St : State .

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