A Partial Evaluation Framework for Order-sorted Equational Programs modulo Axioms*

M. Alpuente[a], A. Cuenca-Ortega[b], S. Escobar[a], J. Meseguer[c]

[a] *VRAIN, Universitat Politècnica de València, Valencia, Spain*
[b] *Universidad de Guayaquil, Guayaquil, Ecuador*
[c] *University of Illinois at Urbana-Champaign, Urbana, IL, USA*

Abstract

Partial evaluation is a powerful and general program optimization technique with many successful applications. Existing PE schemes do not apply to expressive rule-based languages like Maude, CafeOBJ, OBJ, ASF+SDF, and ELAN, which support: 1) rich type structures with sorts, subsorts, and overloading; and 2) equational rewriting modulo various combinations of axioms such as associativity, commutativity, and identity. In this paper, we develop the new foundations needed and illustrate the key concepts by showing how they apply to partial evaluation of expressive programs written in Maude. Our partial evaluation scheme is based on an automatic unfolding algorithm that computes term *variants* and relies on high-performance *order-sorted equational least general generalization* and *order-sorted equational homeomorphic embedding* algorithms for ensuring termination. We show that our partial evaluation technique is sound and complete for convergent rewrite theories that may contain various combinations of associativity, commutativity, and/or identity axioms for different binary operators. We demonstrate the effectiveness of Maude's automatic partial evaluator, Victoria, on several examples where it shows significant speed-ups.

*This work has been partially supported by the EU (FEDER) and the Spanish MCIU under grant RTI2018-094403-B-C32, by Generalitat Valenciana under grant PROMETEO/2019/098, and by NRL under contract number N00173-17-1-G002. Angel Cuenca-Ortega has been supported by the SENESCYT, Ecuador (scholarship program 2013).

Email addresses: alpuente@upv.es (M. Alpuente), angel.cuencao@ug.edu.ec (A. Cuenca-Ortega), sescobar@upv.es (S. Escobar), meseguer@illinois.edu (J. Meseguer)

1. Introduction

Partial evaluation (PE), also known as program specialization, is a semantics-based program transformation technique in which a program is specialized to a part of its input that is known statically (at *specialization* time) [27, 42]. Program specializers employ a very simple transformational technique: the selective *symbolic execution* of the program [65]. The most straightforward application is to produce a new "residual program", which typically runs faster (on the remaining input data) than the original program (on all of its input data), while being guaranteed to behave in the same way. Dramatic savings can be achieved in programs that will be executed many times with few variations of their parameters since many costly computations are performed once and for all during PE. Partial evaluation has currently reached a point where theory and refinements have matured, substantial systems have been developed, and realistic applications can benefit from partial evaluation in a wide range of fields that transcend the optimization of programs by far. Novel applications of partial evaluation include model-driven development, domain-specific language engineering, generic programming, and test-case generation, just to mention a few [22, 26].

Narrowing-driven PE (NPE) [13, 12, 14, 15] is a generic algorithm for the specialization of functional programs that are executed by *narrowing*, a symbolic execution mechanism that extends term rewriting by replacing pattern matching with unification [37, 61]. Essentially, narrowing consists of computing an appropriate substitution for a symbolic program call (input term) in such a way that the program call becomes reducible by a program equation (that is implicitly oriented as a rewrite rule), and then reducing it: both the rewrite rule and the term are instantiated. As in logic programming, narrowing computations can be represented by a (possibly infinite) finitely branching tree. Since narrowing subsumes both rewriting and SLD-resolution, it is complete in the sense of both functional programming (computation of normal forms) and logic programming (computation of answers). The NPE method generalizes the theoretical framework for the partial evaluation of logic programs established in [50, 51] (also known as *partial deduction*, PD) to functional logic programs, with the key concepts being extended to suitably cope with functions and nested function calls (e.g., the *closedness* condition that ensures that all calls that might occur during the execution of the specialized program are covered by the specialized program itself). By combining the functional dimension of narrowing with the power of logical variables and unification, the NPE approach has better opportunities for optimization than the more standard PD (resp. PE) of logic (resp. functional) programs [15].

To the best of our knowledge, partial evaluation has never been investigated in the context of expressive rule-based languages like Maude, CafeOBJ, OBJ,

ASF+SDF, and ELAN, which support: (i) rich type structures with sorts, subsorts, and overloading; and (ii) equational rewriting modulo various combinations of axioms such as associativity (A), commutativity (C), and identity (U). When we consider (i) and (ii) in a contemporary, algebraic (as well as logic) language such as Maude, the key PE ingredients of [14] have to be further generalized to corresponding (order-sorted) *equational* notions (*modulo* axioms): e.g., *equational unfolding, equational closedness, equational embedding, and equational abstraction*; and the associated partial evaluation techniques become more sophisticated and powerful. In this paper, we develop such equational foundations and show how they apply to the partial evaluation of expressive rule-based programs that are written in Maude.

In this paper, we deal with Maude's order-sorted equational theories $(\Sigma, E \uplus B)$ that are decomposed as rewrite theories $(\Sigma, B, \vec{E})$ [53], where B is a set of commonly occurring axioms such as associativity, commutativity, and identity, and $\vec{E}$ is a set of oriented convergent equations, i.e., a set of rules such that rewriting with $\vec{E}$ modulo B transforms every term into a unique irreducible form (see [56]). In addition to rewriting in $(\Sigma, B, \vec{E})$, an order-sorted equational theory $(\Sigma, E \uplus B)$ can also be symbolically executed in Maude with the oriented equations $\vec{E}$ modulo the axioms B by using the "folding variant narrowing strategy" of [36]. This form of narrowing is useful for variant computation and (variant-based) equational unification, and as we show in this paper, for partial evaluation. On the other hand, unification modulo combinations of associativity, commutativity, and identity is used by folding variant narrowing. The main idea[1] of folding variant narrowing is to "fold", by subsumption modulo B, the $(\vec{E}, B)$-narrowing tree that can in practice result in a finite, directed acyclic narrowing graph that symbolically and concisely summarizes the (generally infinite) narrowing search space for $(\Sigma, B, \vec{E})$. When the equational theory $(\Sigma, E \uplus B)$ additionally has the finite variant property [25, 36], a finite complete set of most general $(\vec{E},B)$-variants exists for each term t, where each $(\vec{E},B)$-variant of t consists of a substitution σ and the $(\vec{E},B)$-irreducible form of $t\sigma$.

Let us motivate the power of our technique by reproducing the classical specialization of a parser w.r.t. a given grammar into a very specialized parser [42].

[1] The notion of folding in folding variant narrowing is essentially a *subsumption* notion applied to some leaves of the narrowing tree, so that less general leaves are subsumed (folded into) more general ones. Therefore, this notion is quite different from the classical folding operation of Burstall and Darlington's fold/unfold transformation scheme [21], 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. As for unfolding, following the narrowing-driven partial evaluation approach of [14], we symbolically execute the function calls by using Maude's narrowing strategy for equational theories, i.e., the folding variant narrowing strategy.

The parser is encoded as an equational theory (Maude *functional module*), that contains several language features that are *unknown territory*[2] for state-of-the-art (narrowing-driven) partial evaluation: (i) a hierarchy of sorts that defines the sorts `TSymbol` (terminal symbols) and `NSymbol` (non-terminal symbols) as two subsorts of the sort `Symbol` of all grammar symbols; and (ii) an associative-commutative with identity operator `_;_` for representing grammars (meaning that they are handled as a multiset[3] of productions), together with the operator `__` that is used to represent the input string and has the empty string as a right identity element.

Example 1. *Consider the following equational theory (written in Maude*[4] *syntax) that defines a generic parser*[5] *for languages generated by simple, right regular grammars. We define a free constructor operator* `_|_|_` *to represent the parser configurations, where the first underscore represents the (terminal or non-terminal) symbol being processed, the second underscore represents the current string pending recognition, and the third underscore stands for the considered grammar. We provide two non-terminal symbols,* `init` *and* `S`*, and three terminal symbols,* `0`*,* `1`*, and the finalizing mark* `eps` *(for* ε*, the empty string). These are useful choices for this example, but they can be easily generalized to any terminal and non-terminal symbols by defining a Maude parameterized theory. Parsing a string* `st` *according to a given grammar* Γ *is defined by rewriting the configuration* `(init | st | Γ)` *using the rules of the grammar (right-to-left) to incrementally transform* `st` *until the final, success configuration* `(eps | eps | Γ)` *is reached.*

```
fmod PARSER is
  sorts Symbol NSymbol TSymbol String Production Grammar Parsing .
  subsort Production < Grammar .
  subsort TSymbol < String .
```

[2]Classical PE applies to rewrite theories $(\Sigma, \emptyset, \vec{E})$ where Σ is untyped and there are no axioms ($B = \emptyset$). This is a very special case of the framework provided in this paper for any $(\Sigma, B, \vec{E})$ with Σ being order-sorted and B being any combination of associativity, commutativity, and identity axioms.

[3]Multisets are very common data structures in Maude programming whose advantage is (at least) twofold. Firstly, multisets are easy to define: you just need to declare a binary ACU operator plus a constant (i.e., the identity element) that identifies the empty soup. Secondly, (nondeterministic) element selection in a multiset is automatically performed by Maude's extremely efficient ACU-matching algorithm. Using sets would be a bit more elaborate since the set union operator is ACUI (associative-commutative-identity-idempotent) and ACUI-unification is not directly supported by Maude.

[4]In Maude 2.7.1, only equations with the attribute `variant` are used by the folding variant narrowing strategy.

[5]In [32], a similar parser is encoded by using Maude rewrite rules that more naturally represent the system evolution. However, laying the foundations for the partial evaluation of general Maude rewrite theories is a matter for future research.

```
  subsorts TSymbol NSymbol < Symbol .
  ops 0 1 eps : -> TSymbol .
  ops init S : -> NSymbol .
  op mt : -> Grammar .
  op __ : TSymbol String -> String [right id: eps].
  op _->_ : NSymbol TSymbol -> Production .
  op _->_._ : NSymbol TSymbol NSymbol -> Production .
  op _;_ : Grammar Grammar -> Grammar [assoc comm id: mt] .
  op _|_|_ : Symbol String Grammar -> Parsing .
  var E : TSymbol .
  vars N M : NSymbol .
  var  L : String .
  var G : Grammar .
  eq (N | eps | ( N ->  eps ) ; G)
   =  (eps | eps | ( N ->  eps ) ; G) [variant] .
  eq (N | E L | ( N -> E . M ) ; G)
   = (M | L | ( N -> E . M ) ; G) [variant] .
endfm
```

The general case of the parser is defined by the second equation that, given the configuration `(N | E L | Γ)`*, where* `(E L)` *is the string to be recognized, searches for the grammar production* `(N -> E . M)` *in* Γ *to recognize symbol* `E`*, and proceeds to recognize* `L` *starting from the non-terminal symbol* `M`*. Note that if there are no two grammar productions in* Γ *of the form* `N -> E.`$\mathtt{M}_1$ *and* `N -> E.`$\mathtt{M}_2$*, the equational theory produces unique normal forms for any input term of sort* `Parsing`*, since there is always only one possible application of the second equation. Also note that the combination of subtypes and equational (algebraic) axioms allows for a very compact definition.*

For example, given the following grammar Γ *that generates the language* 0^*1^**:*

```
init -> eps ; init -> 0 . init ; init -> 1 . S ; S -> eps ; S -> 1 . S
```

the initial configuration `(init | 0 0 1 1 |` Γ`)` *is deterministically rewritten modulo axioms as* `(init | 0 0 1 1 |` Γ`)`$\rightarrow$ `(init | 0 1 1 |` Γ`)`$\rightarrow$ `(init | 1 1 |` Γ`)`$\rightarrow$ `(S | 1 |` Γ`)`$\rightarrow$ `(S | eps |` Γ`)`$\rightarrow$ `(eps | eps |` Γ`)`.

We can specialize our parsing program to the productions of the given grammar Γ *by partially evaluating the input term* `(init | L |` Γ`)`*, where* `L` *is a logical variable of sort* `String`*. By applying our partial evaluator, we aim to obtain the specialized parsing equations:*

```
eq init | eps | Γ = eps  | eps| Γ [variant] .
eq init | 0 L | Γ = init | L  | Γ [variant] .
eq init | 1 L | Γ = S    | L  | Γ [variant] .
eq    S | eps | Γ = eps  | eps| Γ [variant] .
eq    S | 1 L | Γ = S    | L  | Γ [variant] .
```

that still recognizes the string `st` *by rewriting the simpler configuration* `(init | st | Γ)` *to the final configuration* `(eps | eps | Γ)`. *As discussed in Section 5, the specialized program shows an impressive speed-up with respect to the original parser.*

Our contribution. This paper focuses on the foundations of our order-sorted equational partial evaluation system, i.e., the core notions, principles, and algorithms. To the best of our knowledge, this is the first partial evaluation framework in the literature for order-sorted equational theories that is able to cope with subsorts, subsort polymorphism, convergent rules (equations), and equational axioms. We base our partial evaluator on a suitably generalized version of the general NPE procedure of [14], which is parametric w.r.t. the *unfolding rule* used to construct finite computation trees and also w.r.t. an *abstraction operator* that is used to guarantee that only finitely many expressions are evaluated. For unfolding, we use *folding variant narrowing* [36], which is an optimal narrowing strategy for convergent equational theories that computes *most general variants* modulo algebraic axioms and is efficiently implemented in Maude. For the abstraction, we rely on the *order-sorted equational least general generalization* recently investigated in [8, 7]. As in [13], we follow the on-line approach to PE that makes control decisions about specialization on the fly, which is simpler to describe and offers better opportunities for powerful automated strategies than off-line partial evaluation [42, 24], where decisions are made before specialization by using abstract data descriptions that are represented as program annotations.

Related work. Program specialization has been investigated within different programming paradigms and applied to a wide variety of languages. Among the vast literature on program specialization, the partial evaluation of functional logic programs is the closest to our work. For the (rewriting-based) functional logic language Escher, a partial evaluator was described in [45]. The work by Darlington and Pull [28] is the most clear predecessor of narrowing-driven partial evaluation. They proposed the use of narrowing as an alternative to the combination of instantiation and unfolding—in the sense of Burstall and Darlington [21]—to perform partial evaluation. Their approach yielded a partial evaluator for the functional language HOPE extended with unification. For the functional logic language Curry [40], a partial evaluator based on needed narrowing was first proposed in [2]. The most recent partial evaluator for functional logic programs is described in [41],

which is able to deal with Curry programs that may contain non-deterministic operations. For a recent discussion regarding the practical partial evaluation of mainstream languages such as JavaScript, Ruby, and R, see [66]. Obviously, none of these PE systems can deal with the salient features (subtype polymorphism and computing modulo axioms) considered in this work.

Comparison with our previous work [5]. The partial evaluation of equational programs modulo axioms was first investigated in an earlier conference paper [5]. Its development relied on two key notions: (order-sorted) homeomorphic embedding modulo equational axioms and (order-sorted) equational least general generalization (which is used for the formulation of equational abstraction). The novel contributions with respect to [5] are as follows:

1. The homeomorphic embedding modulo equational axioms $\trianglelefteq_B$ of [5] did not scale up to realistic problems. Furthermore, the formalization in [5] did not consider sorts and subsorts, so preposterous embedding tests such as `X:Bool` $\trianglelefteq_B$ `(0 + suc(N:Nat))` succeed. The equational abstraction of [5] was formalized by relying on the rather expensive equational least general generalization calculus of [8], which significantly degraded the partial evaluator performance. In this article, we provide a more effective formulation of both notions based on our recent results in [7] and [4], which increases the specializing power of Victoria and has significantly boosted its performance.
2. A fuller treatment of all aspects is given, including more examples and detailed explanations of the key notions, together with full formal proofs of all technical results.
3. In line with the extended definitions, we have correspondingly extended our technical results for dealing with sorts and subsorts in a finer way.
4. The experimental work has been improved in two ways:
   (a) The prototype tool itself has been advanced so that now it is a fully automated robust system that scales up to much more complex partial evaluation problems than the semi-automated, preliminary prototype in [5]. The new system relies on the high-performance implementation of the order-sorted homeomorphic embeddding modulo axioms of [7] and the order-sorted least general generalization algorithm of [4] so that it runs up to six orders of magnitude faster than the previous implementation in [5].
   (b) We experimentally evaluate the current prototype on more ambitious partial evaluation problems, including the implementation of a complete interpreter for an imperative language dealing with loops, conditional statements, and arithmetic expressions in Section 4. By using

Victoria the interpreter is automatically improved without introducing any ad-hoc primitives to the language semantics to guide the interpreter specialization.

Plan of the paper. The rest of the paper is organized as follows. In Section 2, we recall some necessary notions about order-sorted equational theories and narrowing in the Maude language. Section 3 formalizes our partial evaluation scheme by generalizing to the order-sorted and modulo axioms setting the key ingredientes of NPE, namely unfolding (based on narrowing), closedness, homeomorphic embedding, and abstraction (based on generalization). Section 4 illustrates the proposed methodology by describing several specializations of the interpreter of an imperative programming language. Section 5 presents some experiments with the partial evaluator Victoria that implements our technique. The system demonstrates the usefulness of our approach, with some specialized programs running up to two orders of magnitude (100 times) faster than the original one. In Section 6, we draw some conclusions and outline directions for our future work. The proofs of all of technical results are given in Appendix A.

2. Preliminaries

Let us recall some key concepts of order-sorted, rewriting logic theories [53] and equational unification [16]. We consider an order-sorted signature[6] $\Sigma{=}(\Sigma, \mathsf{S}, \leq)$ that consists of a poset of sorts $(\mathsf{S}, \leq)$ and an $\mathsf{S}^* \times \mathsf{S}$-indexed family of sets $\Sigma = \{\Sigma_{\mathsf{s}_1 \ldots \mathsf{s}_n, \mathsf{s}}\}_{(\mathsf{s}_1 \ldots \mathsf{s}_n, \mathsf{s}) \in \mathsf{S}^* \times \mathsf{S}}$ of function symbols. The poset $(\mathsf{S}, \leq)$ of sorts for Σ is partitioned into equivalence classes $C_1, \ldots, C_n$ (called *connected components*) by the equivalence relation $(\leq \cup \geq)^+$. Throughout this paper, Σ is assumed to be *preregular*, so each term t has a least sort, denoted $ls(t)$ (see [38]). Σ is also assumed to be *kind-complete* [55], that is, for each sort $\mathsf{s} \in \mathsf{S}$, its connected component in the poset $(\mathsf{S}, \leq)$ has a top sort under $\leq$, denoted $[\mathsf{s}]$ and called the connected component's *kind*, and for each function symbol $f \in \Sigma_{\mathsf{s}_1 \ldots \mathsf{s}_n, \mathsf{s}}$, there is also an $f \in \Sigma_{[\mathsf{s}_1] \ldots [\mathsf{s}_n], [\mathsf{s}]}$. An order-sorted signature can always be extended to be kind-complete [55]. Maude automatically checks preregularity and adds a new "kind" sort $[\mathsf{s}]$ at the top of the connected component of each sort $\mathsf{s} \in \mathsf{S}$ specified by the user and automatically lifts each operator to the kind level. For technical reasons, it is useful to assume that Σ has no ad-hoc overloading[7]. However, this assumption entails no real loss of

[6] This abuse of language of using Σ for both the triplet and the ranked set of function symbols is useful, and is in fact the notation we use later.

[7] Given the overloaded operator $f : \mathsf{s}_1 \ldots \mathsf{s}_m \longrightarrow \mathsf{s}_0$ and $f : \mathsf{s}'_1 \ldots \mathsf{s}'_n \longrightarrow \mathsf{s}'_0$, 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.

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 $f : \mathsf{s}_1 \ldots \mathsf{s}_n \longrightarrow \mathsf{s}$ and $f : \mathsf{s}'_1 \ldots \mathsf{s}'_n \longrightarrow \mathsf{s}'$ of a n-ary function symbol f, if s_i and s'_i are in the same connected component of $(\mathsf{S}, \leq)$ 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.

We assume an S-sorted family $\mathscr{X} = \{\mathscr{X}_\mathsf{s}\}_{\mathsf{s}\in\mathsf{S}}$ of disjoint variable sets. $\mathscr{T}_{\Sigma,\mathsf{s}}(\mathscr{X})$ and $\mathscr{T}_{\Sigma,\mathsf{s}}$ denote the sets of terms and ground terms of sort s, respectively. Note that $\mathsf{s} < \mathsf{s}'$ (s is a subsort of s') implies the set of terms of sort s are a subset of the set of terms of sort s', i.e., $\mathscr{T}_{\Sigma,\mathsf{s}}(\mathscr{X}) \subseteq \mathscr{T}_{\Sigma,\mathsf{s}'}(\mathscr{X})$. We (ambiguously) write $\mathscr{T}_\Sigma(\mathscr{X})$ and $\mathscr{T}_\Sigma$ for *both* the corresponding term algebras and for the underlying sets of terms, i.e., $\mathscr{T}_\Sigma(\mathscr{X}) = \cup_{\mathsf{s}\in\mathsf{S}}\mathscr{T}_{\Sigma,\mathsf{s}}(\mathscr{X})$ and $\mathscr{T}_\Sigma = \cup_{\mathsf{s}\in\mathsf{S}}\mathscr{T}_{\Sigma,\mathsf{s}}$. Throughout this paper we assume that $\mathscr{T}_{\Sigma,\mathsf{s}} \neq \emptyset$ for every sort s because this affords a simpler deduction system. The set of variables occurring in a term t is denoted by $Var(t)$. In order to simplify the presentation, we often disregard sorts when no confusion can arise. Let $\rightarrow \subseteq A \times A$ be a binary relation on a set A. We denote its transitive closure by $\rightarrow^+$, and its reflexive and transitive closure by $\rightarrow^*$. A sequence of syntactic objects $o_1, \ldots, o_n$ is denoted by $\bar{o}_n$.

A *position* p in a term t is represented by a sequence of natural numbers (Λ denotes the empty sequence, i.e., the root position). Positions are ordered by the *prefix* ordering: $p \leq q$, if $\exists w$ such that $p.w = q$. Given a term t, we let $Pos(t)$ and $Pos_\Sigma(t)$ respectively denote the set of positions and the set of non-variable positions of t (i.e., positions where a variable does not occur). The expression $t_{|p}$ denotes the *subterm* of t at position p, and $t[u]_p$ denotes the result of *replacing the subterm* $t_{|p}$ by the term u at position p.

A *substitution* σ is a sorted mapping from a finite subset of $\mathscr{X}$ to $\mathscr{T}_\Sigma(\mathscr{X})$. Substitutions are written as $\sigma = \{X_1 \mapsto t_1, \ldots, X_n \mapsto t_n\}$ where the domain of σ is $Dom(\sigma) = \{X_1, \ldots, X_n\}$ and the set of variables introduced by terms $t_1, \ldots, t_n$ is written $Ran(\sigma)$. The identity substitution is denoted id. Substitutions are homomorphically extended to $\mathscr{T}_\Sigma(\mathscr{X})$. The application of a substitution σ to a term t is denoted by $t\sigma$. The restriction of σ to a set of variables $V \subset \mathscr{X}$ is denoted $\sigma_{|V}$; sometimes we write $\sigma_{|t_1,\ldots,t_n}$ to denote $\sigma_{|V}$ where $V = Var(t_1) \cup \cdots \cup Var(t_n)$. Composition of two substitutions is denoted by $\sigma\sigma'$ so that $t(\sigma\sigma') = (t\sigma)\sigma'$.

A Σ-*equation* is an unoriented pair $t = t'$, where $t, t' \in \mathscr{T}_{\Sigma,\mathsf{s}}(\mathscr{X})$ for some sort $\mathsf{s} \in \mathsf{S}$. Given Σ and a set $\mathscr{E}$ of Σ-equations, order-sorted equational logic induces a congruence relation $=_\mathscr{E}$ on $\mathscr{T}_\Sigma(\mathscr{X})$ (see [54]). An *equational theory* $(\Sigma, \mathscr{E})$ is a

Otherwise, the overloading of f is called ad-hoc.

pair with Σ an order-sorted signature and $\mathscr{E}$ a set of Σ-equations. We omit Σ when no confusion can arise.

A term t is more (or equally) general than t' modulo $\mathscr{E}$, denoted by $t \leq_{\mathscr{E}} t'$, if there is a substitution γ such that $t' =_{\mathscr{E}} t\gamma$. A substitution θ is more (or equally) general than 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 $x \in V, x\sigma =_{\mathscr{E}} x\theta\gamma$. We also define $t \simeq_{\mathscr{E}} t'$ iff $t \leq_{\mathscr{E}} t'$ and $t' \leq_{\mathscr{E}} t$; and similarly $\theta \simeq_{\mathscr{E}} \sigma$.

An *$\mathscr{E}$-unifier* for a Σ-equation $t = t'$ is a substitution σ such that $t\sigma =_{\mathscr{E}} t'\sigma$. $CSU_{\mathscr{E}}(t = t')$ denotes a *complete* set of unifiers for the equation $t = t'$ modulo $\mathscr{E}$, so that any $\mathscr{E}$-unifier is an $\mathscr{E}$-instance of one of them. An $\mathscr{E}$-unification algorithm is *complete* if for any equation $t = t'$ it generates a complete set of $\mathscr{E}$-unifiers. Note that this set needs not be finite. A unification algorithm is said to be *finitary* if it always terminates. Note that a complete and finitary $\mathscr{E}$-unification algorithm may not exist even if a complete and finite set of $\mathscr{E}$-unifiers exists.

A *rewrite theory* is a triple $\mathscr{R} = (\Sigma, \mathscr{E}, R)$, where $(\Sigma, \mathscr{E})$ is the equational theory modulo which we rewrite and R is a set of rewrite rules. Rules are of the form $l \to r$ where terms $l, r \in \mathscr{T}_{\Sigma,\mathsf{s}}(\mathscr{X})$ for some sort s are respectively called the *left-hand side* (or *lhs*) and the *right-hand side* (or *rhs*) of the rule and $Var(r) \subseteq Var(l)$. The set R of rules is required to be *sort-decreasing*, i.e., for each $l \to r$ in R, and for each well-sorted substitution σ, $ls(l\sigma) \geq ls(r\sigma)$.

We define the *one-step rewrite relation* on $\mathscr{T}_{\Sigma}(\mathscr{X})$ for the set of rules R as follows: $t \to_R t'$ if there is a position $p \in Pos(t)$, a rule $l \to r$ in R, and a substitution σ such that $t_{|p} = l\sigma$ and $t' = t[r\sigma]_p$. The relation $\to_{R/\mathscr{E}}$ for rewriting modulo $\mathscr{E}$ is defined as $=_{\mathscr{E}} \circ \to_R \circ =_{\mathscr{E}}$. A term t is called *$R/\mathscr{E}$-irreducible* iff there is no term u such that $t \to_{R/\mathscr{E}} u$. A substitution σ is $R/\mathscr{E}$-irreducible if, for every $x \in \mathscr{X}$, $x\sigma$ is $R/\mathscr{E}$-irreducible. We say that the relation $\to_{R/\mathscr{E}}$ is *terminating* if there is no infinite sequence $t_1 \to_{R/\mathscr{E}} t_2 \to_{R/\mathscr{E}} \cdots t_n \to_{R/\mathscr{E}} t_{n+1} \cdots$. We say that the relation $\to_{R/\mathscr{E}}$ is *confluent* if whenever $t \to^*_{R/\mathscr{E}} t'$ and $t \to^*_{R/\mathscr{E}} t''$, there exists a term t''' such that $t' \to^*_{R/\mathscr{E}} t'''$ and $t'' \to^*_{R/\mathscr{E}} t'''$. A rewrite theory $(\Sigma, \mathscr{E}, R)$ is *convergent* if R is sort-decreasing and the relation $\to_{R/\mathscr{E}}$ is confluent and terminating. In a convergent order-sorted rewrite theory, for each term $t \in \mathscr{T}_{\Sigma}(\mathscr{X})$, there is a unique (up to $\mathscr{E}$-equivalence) $R/\mathscr{E}$-irreducible term t' that can be obtained by rewriting t to $R/\mathscr{E}$-irreducible or *normal* form, which is denoted by $t \to^!_{R/\mathscr{E}} t'$, or $t{\downarrow_{R/\mathscr{E}}}$ when t' is not relevant. For each $x \in Dom(\sigma)$, $\sigma{\downarrow_{R/\mathscr{E}}}$ is defined as $(\sigma{\downarrow_{R/\mathscr{E}}})(x) = \sigma(x){\downarrow_{R/\mathscr{E}}}$. A substitution σ is $R/\mathscr{E}$-irreducible (normalized) iff $x\sigma$ is so for each $x \in Dom(\sigma)$. For a set Q of terms, we denote by $Q{\downarrow_{R/\mathscr{E}}}$ the set of normal forms of the terms in Q.

Since $\mathscr{E}$-congruence classes can be infinite, $\to_{R/\mathscr{E}}$-reducibility is undecidable

in general. Therefore, $R/\mathscr{E}$-rewriting is usually implemented [44] by $R,\mathscr{E}$-rewriting. We define the relation $\rightarrow_{R,\mathscr{E}}$ on $\mathscr{T}_\Sigma(\mathscr{X})$ by $t \rightarrow_{p,R,\mathscr{E}} t'$ (or simply $t \rightarrow_{R,\mathscr{E}} t'$) iff there is a non-variable position $p \in Pos_\Sigma(t)$, a rule $l \rightarrow r$ in R, and a substitution σ such that $t_{|p} =_\mathscr{E} l\sigma$ and $t' = t[r\sigma]_p$. To ensure completeness of $R,\mathscr{E}$-rewriting w.r.t. $R/\mathscr{E}$-rewriting, we require *strict coherence*, ensuring that $=_\mathscr{E}$ is a bisimulation for $R,\mathscr{E}$-rewriting [56]: for any Σ-terms u,u',v if $u =_\mathscr{E} u'$ and $u \rightarrow_{R,\mathscr{E}} v$, then there exists a term v' such that $u' \rightarrow_{R,\mathscr{E}} v'$ and $v =_\mathscr{E} v'$. Note that, assuming $\mathscr{E}$-matching is decidable, $\rightarrow_{R,\mathscr{E}}$ is decidable and notions such as confluence, termination, irreducible term, and normalized substitution are defined for $\rightarrow_{R,\mathscr{E}}$ straightforwardly [56]. It is worth noting that Maude automatically provides strict $\mathscr{E}$-coherence completion 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.

2.1. 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 [34] but are instead handled implicitly by working with congruence classes of terms. This is why an equational theory is often decomposed into a disjoint union $\mathscr{E} = E \uplus B$, 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 consists of (possibly conditional) 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 $(\Sigma, B, \vec{E})$ is 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* of the equational theory $(\Sigma, \mathscr{E})$ as follows.

Definition 1 (Decomposition [35]**).** *Let $(\Sigma, \mathscr{E})$ be an order-sorted equational theory. We call $(\Sigma, B, \vec{E})$ a* decomposition *of $(\Sigma, \mathscr{E})$ if $\mathscr{E} = E \uplus B$ and $(\Sigma, B, \vec{E})$ 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 $t\sigma \in \mathscr{T}_{\Sigma,s}(\mathscr{X})$ iff $t'\sigma \in \mathscr{T}_{\Sigma,s}(\mathscr{X})$. 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 strictly coherent modulo B, and* sort-decreasing.

We often abuse notation and say that $(\Sigma, B, \vec{E})$ is a decomposition of an order-sorted equational theory $(\Sigma, \mathscr{E})$ even if $\mathscr{E} \neq E \uplus B$ but E is instead the explicitly extended B-coherent completion of a set E' such that $\mathscr{E} = E' \uplus B$.

Given the rewrite theory $(\Sigma, B, \vec{E})$, it is common to split the signature Σ into two disjoint sets: defined symbols and constructor symbols. *Defined symbols* are defined as $\mathscr{D} = \{f \in \Sigma \mid \exists f(t_1, \ldots, t_n) \to r \in \vec{E}\}$, and *constructors*[8] are defined as $\mathscr{C} = \Sigma \setminus \mathscr{D}$.

In the following, we often consider rewrite theories (Σ, B, R) that are a decomposition of an order-sorted equational theory, so that $R = \vec{E}$.

2.2. Narrowing in Rewriting Logic

Narrowing generalizes term rewriting by allowing free variables in terms (as in logic programming) and by performing unification (at non-variable positions) instead of matching in order to (non-deterministically) reduce the term. Function definition and evaluation are thus embedded within a symbolic logical framework and features such as existentially quantified variables, unification, and function inversion become available.

Definition 2 ((R,B)-narrowing [58]**).** *Let $\mathscr{R} = (\Sigma, B, R)$ be an order-sorted rewrite theory. The* (R,B)-narrowing *relation on $\mathscr{T}_\Sigma(\mathscr{X})$ is defined as $t \leadsto_{\sigma,R,B} t'$ (or just $t \leadsto_\sigma t'$) if there exist $p \in Pos_\Sigma(t)$, a (renamed apart*[9]*) rule $l \to r$ in R, and a B-unifier σ of $t_{|p}$ and l such that $t' = (t[r]_p)\sigma$. The narrowing step $t \leadsto_{\sigma,R,B} t'$ is also called a* (R,B)-narrowing step*. A term t is* (R,B)-narrowable *if there exist σ and t' such that $t \leadsto_{\sigma,R,B} t'$. Given the narrowing sequence $\alpha : (t_0 \leadsto_{\theta_1} t_1 \cdots \leadsto_{\theta_n} t_n)$, the computed substitution of α is $\theta = (\theta_1 \ldots \theta_n)_{|Var(t_0)}$ and we may write $t_0 \leadsto^n_\theta t_n$.*

[8] In what follows, we assume for simplicity that the notions of defined and constructor symbol are independent of the symbol's typing. However, in an order-sorted setting this need not be the case. For a simple example, consider that for a sort `Nat` of natural numbers with constants 0 and 1, the addition function + can be declared as a *constructor* modulo the axioms B of associativity (A), commutativity (C), and 0 as an identity (U). However, when we add the supersort `Nat < Int` of integers, the definition of integer addition +, besides being also ACU, needs additional equations in $\vec{E}$ and therefore is not a constructor. We assume throughout that, given a symbol f in Σ, all its typings are either constructors or they are defined. The more general case illustrated by the above example for number addition is left for future work.

[9] The renaming is chosen to ensure that every substitution is idempotent, i.e., σ satisfies $Dom(\sigma) \cap Ran(\sigma) = \emptyset$ so that $(t\sigma)\sigma = t\sigma$.

Since (R,B)-narrowing has quite a large search space, suitable strategies are needed to improve the efficiency of narrowing by getting rid of useless computations.

First, we define the notion of a narrowing strategy. Given a (R,B)-narrowing sequence $\alpha : (t_0 \leadsto_{\theta_1} t_1 \cdots \leadsto_{\theta_n} t_n)$, we denote by α_i the narrowing sequence $\alpha_i : (t_0 \leadsto_{\theta_1} t_1 \cdots \leadsto_{\theta_i} t_i)$, which is a prefix of α. Given an order-sorted rewrite theory $\mathscr{R}$, we denote by $Full_{\mathscr{R}}(t)$ the (possibly infinite) set of all (R,B)-narrowing sequences stemming from t.

Definition 3 (Narrowing Strategy). *A* narrowing strategy *is a function of two arguments: a rewrite theory $\mathscr{R} = (\Sigma, B, R)$ and a term $t \in \mathscr{T}_\Sigma(\mathscr{X})$, which we denote by $\mathscr{S}_{\mathscr{R}}(t)$, such that $\mathscr{S}_{\mathscr{R}}(t) \subseteq Full_{\mathscr{R}}(t)$. We require $\mathscr{S}_{\mathscr{R}}(t)$ to be prefix closed, i.e., for each narrowing sequence $\alpha \in \mathscr{S}_{\mathscr{R}}(t)$ of length n, and each $i \in \{1,\ldots,n\}$, we also have $\alpha_i \in \mathscr{S}_{\mathscr{R}}(t)$.*

Narrowing strategies for rewrite theories that are complete (i.e., for every solution, it computes a more general answer) under suitable conditions have been investigated in [58, 62].

Example 2. *The equational theory for exclusive-or has a decomposition into $\vec{E}$ consisting of the (implicitly oriented) equations* (1)-(3) *below, and B the associativity and commutativity (AC) axioms for symbol $\oplus$:*

$$X \oplus 0 = X \quad (1) \qquad X \oplus X = 0 \quad (2) \qquad X \oplus X \oplus Y = Y \quad (3)$$

Note that equations (1)-(2) *are not strictly AC-coherent, but adding equation* (3) *is sufficient to recover that property (see [63, 33]).*

Given the term $t = X \oplus Y$, the following $(\vec{E},B)$-narrowing steps can be proved (only the bindings for the variables X and Y of the input term are shown and, by applying the renaming apart technique, variables X and Y of the equations (1)-(3) *are respectively renamed as X' and Y')*

$X \oplus Y \leadsto_{\phi_1} X'$ *using* $\phi_1 = \{X \mapsto 0, Y \mapsto X'\}$ *and Equation* (1)
$X \oplus Y \leadsto_{\phi_2} X'$ *using* $\phi_2 = \{X \mapsto X', Y \mapsto 0\}$ *and Equation* (1)
$X \oplus Y \leadsto_{\phi_3} 0$ *using* $\phi_3 = \{X \mapsto X', Y \mapsto X'\}$ *and Equation* (2)
$X \oplus Y \leadsto_{\phi_4} Y'$ *using* $\phi_4 = \{X \mapsto Y' \oplus X', Y \mapsto X'\}$ *and Equation* (3)
$X \oplus Y \leadsto_{\phi_5} Y'$ *using* $\phi_5 = \{X \mapsto X', Y \mapsto Y' \oplus X'\}$ *and Equation* (3)
$X \oplus Y \leadsto_{\phi_6} U \oplus Y'$ *using* $\phi_6 = \{X \mapsto X' \oplus U, Y \mapsto X' \oplus Y'\}$ *and Equation* (3)

As explained above, in order to provide a finitary and complete unification algorithm for a decomposition $(\Sigma, B, \vec{E})$, two narrowing strategies are defined in [36]: *variant narrowing* and *folding variant narrowing*. After introducing *term variants* [25], these narrowing strategies are formalized in the following sections.

2.3. Term Variants

Intuitively, an $(\vec{E},B)$-*variant* of a term t is the $(\vec{E},B)$-irreducible form of an *instance* $t\sigma$ of t. That is, the variants of t are all of the possible $(\vec{E},B)$-irreducible terms to which instances of t evaluate. Note that variant terms are normalized.

Definition 4 (Term Variant [25]**).** *Given a term t and an equational theory $(\Sigma, E \uplus B)$ with a decomposition $(\Sigma, B, \vec{E})$, we say that (t', θ) is a* variant *of t if $t' =_B (t\theta)\downarrow_{\vec{E},B}$, where $Dom(\theta) \subseteq Var(t)$ and $Ran(\theta) \cap Var(t) = \emptyset$.*

Example 3. *Consider the following basic specification for the addition of natural numbers without axioms:*

$$0 + M = M$$
$$s(N) + M = s(N + M)$$

The set of variants for the term $X + 0$ is infinite, since we have $(0, \{X \mapsto 0\})$, $(s(0), \{X \mapsto s(0)\})$, ..., $(s^k(0), \{X \mapsto s^k(0)\})$. Analogously, the variants of the term $0 + Y$ are $(0, \{Y \mapsto 0\})$, $(s(0), \{Y \mapsto s(0)\})$, ..., $(s^k(0), \{Y \mapsto s^k(0)\})$.

In order to capture when a newly generated variant is subsumed by a previously generated one, we introduce the notion of *variant preordering with normalization*.

Definition 5 (More General Variant [36]**).** *Given a decomposition $(\Sigma, B, \vec{E})$ and two term variants $(t_1, \theta_1), (t_2, \theta_2)$ of a term t, we write $(t_1, \theta_1) \leq_{\vec{E},B} (t_2, \theta_2)$, meaning (t_1, θ_1) is a more general variant of t than (t_2, θ_2), iff there is a substitution ρ such that $(\theta_1\rho)_{|Var(t)} =_B (\theta_2\downarrow_{\vec{E},B})_{|Var(t)}$ and $t_1\rho =_B t_2$.*

Example 4. *The term $N + M$ has an infinite set of most general variants in the theory of Example 3, since we have $(M, \{N \mapsto 0\})$, $(s(M), \{N \mapsto s(0)\})$, ..., $(s^k(M), \{N \mapsto s^k(0)\})$. However, note that the variant $(0, \{N \mapsto 0, M \mapsto 0\})$ is subsumed by $(M, \{N \mapsto 0\})$ and is therefore discarded from the set of most general variants. The set of most general variants of the term $0 + M$ is finite and is $\{(M, id)\}$.*

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. The specification of natural numbers of Example 3 is not a finite variant theory, since the term $N + M$ has an infinite number of most general variants, as shown in Example 4. The equational theory for exclusive-or of Example 2 is a finite variant theory as it is the following theory for Boolean expressions.

Example 5. *Consider the following theory that declares the two Boolean constants* `true` *and* `false`*. The key things to note are the special attributes* `assoc` *and* `comm`*, meaning that the infix operators "*`and`*" and "*`or`*" obey associativity and commutativity axioms:*

```
fmod BOOL is
  sort Bool .
  ops true false : -> Bool .
  op not : Bool -> Bool .
  ops _and_ _or_ : Bool Bool -> Bool [assoc comm] .
  vars X Y : Bool .
  eq not(true) = false [variant] .
  eq not(false) = true [variant] .
  eq X and true = X [variant] .
  eq X and false = false [variant] .
  eq X or true = true [variant] .
  eq X or false = X [variant] .
endfm
```

There are five most general variants modulo AC for "`X and Y`*", which are:* $(\texttt{X and Y}, id), (\texttt{Y}, \{\texttt{X} \mapsto \texttt{true}\}), (\texttt{X}, \{\texttt{Y} \mapsto \texttt{true}\}), (\texttt{false}, \{\texttt{X} \mapsto \texttt{false}\}), (\texttt{false}, \{\texttt{Y} \mapsto \texttt{false}\})$*. Similarly, there are five most general variants for "*`X or Y`*".*

It is generally undecidable whether an equational theory has the FVP [18]; a semi-decision procedure is given in [23, 57] that works well in practice, and another technique based on the dependency pair framework is given in [36]. The procedure in [23] is implemented in [11] and 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 [23].

2.4. The variant narrowing strategy

Given a decomposition $(\Sigma, B, \vec{E})$, applying narrowing without any restriction can be very wasteful due to two main sources: (i) for axioms B such as associativity-commutativity, the number of B-unifiers of an equation can be quite large; and (ii) if we narrow a term in all possible positions, the narrowing tree may grow in an explosive way. Let us first motivate the variant narrowing strategy with two ideas. First, for computing variants in a decomposition we are only interested in *normalized terms* and *normalized substitutions*, so we can restrict our interest to narrowing derivations that provide only normalized substitutions and end in normalized terms, whereas the unrestricted narrowing formalized in Definition 2 does not ensure that.

Example 6. *Continuing with Example 2, due to the prolific AC-unification algorithm there are some redundant narrowing steps with non-normalized substitutions, such as*

$X \oplus Y \leadsto_{\phi_7} X' \oplus U$ *using* $\phi_7 = \{X \mapsto X' \oplus 0, Y \mapsto U\}$ *and Equation* (1)
$X \oplus Y \leadsto_{\phi_8} U \oplus X'$ *using* $\phi_8 = \{X \mapsto U, Y \mapsto 0 \oplus X'\}$ *and Equation* (1)
$X \oplus Y \leadsto_{\phi_9} Y'$ *using* $\phi_9 = \{X \mapsto X' \oplus X', Y \mapsto Y'\}$ *and Equation* (3)
$X \oplus Y \leadsto_{\phi_{10}} Y'$ *using* $\phi_{10} = \{X \mapsto Y', Y \mapsto X' \oplus X'\}$ *and Equation* (3)
$X \oplus Y \leadsto_{\phi_{11}} Y' \oplus U$ *using* $\phi_{11} = \{X \mapsto X' \oplus X' \oplus Y', Y \mapsto U\}$ *and Equation* (3)
$X \oplus Y \leadsto_{\phi_{12}} U \oplus Y'$ *using* $\phi_{12} = \{X \mapsto U, Y \mapsto X' \oplus X' \oplus Y'\}$ *and Equation* (3)

For instance, note that the narrowing step with substitution ϕ_9 *is not needed because the same effect is achieved with the normalized substitution* ϕ_1*. Indeed, the* `vu-narrow` *command of Maude 3.0 [31], which performs full (i.e., unrestricted) narrowing with rules modulo axioms, (non-deterministically) computes* 28 *different one-step narrowing derivations from the term* $X \oplus Y$*. When we consider narrowing sequences instead of single steps, we can easily get a combinatorial explosion, since we have another* 28 *different one-step narrowing derivations after any of the following ones (for simplicity, the three equation renamings have been chosen to produce variables* $Z_1 \oplus Z_2$*):* $X \oplus Y \leadsto_{\phi_6} Z_1 \oplus Z_2$*,* $X \oplus Y \leadsto_{\phi_8} Z_1 \oplus Z_2$*, or* $X \oplus Y \leadsto_{\phi_{11}} Z_1 \oplus Z_2$*. Also, there are many infinite narrowing sequences, such as the one repeating substitution* ϕ_6 *again and again:* $X \oplus Y \leadsto_{\phi_6} Z_1 \oplus Z_2 \leadsto_{\phi'_6} Z'_1 \oplus Z'_2 \leadsto_{\phi''_6} Z''_1 \oplus Z''_2 \leadsto \cdots$ *where* $\phi'_6 = \{Z_1 \mapsto U' \oplus Z'_1, Z_2 \mapsto U' \oplus Z'_2\}$ *and* $\phi''_6 = \{Z'_1 \mapsto U'' \oplus Z''_1, Z'_2 \mapsto U'' \oplus Z''_2\}$.

Our second idea is to give priority to *most general* narrowing steps, instead of dealing with more instantiated ones, and to select one and only one narrowing step among those having the same generality, following a *don't care* approach. This has three implications. The first is that the most general narrowing steps are rewriting steps (if any), and thus any (deterministic) rewrite step should be taken before exploring (possibly non-deterministic) narrowing steps. This resembles the optimization of narrowing known as *normalizing narrowing* (see, e.g., [39]). Thanks to convergence modulo B, as soon as a rewrite step $\rightarrow_{\vec{E},B}$ is enabled in a term that also has narrowing steps $\leadsto_{\vec{E},B}$, such a rewrite step is always taken before any further narrowing steps are applied. The idea of normalizing terms before any narrowing step is taken is consistent with the implementation of rewriting logic [63], where deterministic rewrite steps (with equations) are given priority w.r.t. more expensive, non-deterministic rewrite steps (with rules). The second implication is that variant narrowing goes much further than just giving priority to rewrite steps by filtering out all narrowing steps that do not compute most general substitutions.

Namely, given two narrowing steps $t \leadsto_{\sigma_1,\vec{E},B} t_1$ and $t \leadsto_{\sigma_2,\vec{E},B} t_2$ in a decomposition $(\Sigma, B, \vec{E})$ such that $\sigma_1 \leq_B \sigma_2$, we can safely disregard the narrowing step using σ_2 without losing completeness (c.f. [36, Theorem 4]). The third implication is that we can pack together, in the same equivalence class, all narrowing steps with equally general substitutions and select just one of them as the class representative, thanks to convergence modulo B (see [36] for further details).

Example 7. *The* `vu-narrow` *command of Maude 3.0 [31] computes* 12 *different one-step narrowing derivations from the term* $X \oplus Y \oplus X \oplus Y$ *in the equational theory of Example 2, whereas variant narrowing recognizes that the term is not yet normalized, e.g.,* $X \oplus Y \oplus X \oplus Y \to 0$ *(by using Equation 2), and such a rewriting step is more general than any other narrowing step from* $X \oplus Y \oplus X \oplus Y$*. Thus, such narrowing steps can be disregarded by just choosing to rewrite the term. Note that there are two other rewrite steps* $X \oplus Y \oplus X \oplus Y \to Y \oplus Y$ *(by using Equation 2) and* $X \oplus Y \oplus X \oplus Y \to X \oplus X$ *(by using Equation 2), and equational rewriting in Maude will choose the one that rewrites the maximal* $\oplus$*-term possible due to implicit coherence extensions for rewriting (see [63, 33]).*

On the other hand, if we add a new (redundant) equation

$$X \oplus X \oplus Z \oplus Z \oplus Y = Y \tag{4}$$

to the equational theory of Example 2, the `vu-narrow` *command of Maude 3.0 computes* 173 *different one-step narrowing derivations for the more general term* $X \oplus Y$*. One of them is*

$$X \oplus Y \leadsto_{\mu} Y'' \quad \begin{array}{l} \textit{using the B-unifier} \\ \mu = \{X \mapsto Y'' \oplus X'' \oplus Z'', Y \mapsto X'' \oplus Z''\} \\ \textit{and Equation (4)} \end{array}$$

which is an AC-instance of the narrowing step using substitution ϕ_4 *shown above:*

$$X \oplus Y \leadsto_{\phi_4} Y' \quad \textit{using } \phi_4 = \{X \mapsto Y' \oplus X', Y \mapsto X'\} \textit{ and Equation (3)}$$

Variant narrowing does discard the less general narrowing step with μ*, keeping only the more general narrowing step with* ϕ_4*.*

These optimizations are formalized as follows. First, a preorder between narrowing steps is introduced that defines when a narrowing step is more general than another narrowing step.

Definition 6 (Preorder and Equivalence of Narrowing Steps [36]**).** *Given a decomposition* $(\Sigma, B, \vec{E})$*, consider two narrowing steps* $\alpha_1 : t \leadsto_{\sigma_1,\vec{E},B} s_1$ *and* $\alpha_2 : t \leadsto_{\sigma_2,\vec{E},B} s_2$*. Let* $V = Var(t)$*. We write* $\alpha_1 \preceq_B \alpha_2$ *if* $\sigma_1 \leq_B \sigma_2[V]$ *and* $\alpha_1 \prec_B \alpha_2$ *if* $\sigma_1 <_B \sigma_2[V]$ *(i.e.,* σ_1 *is strictly more general than* σ_2 *on* V*). We write* $\alpha_1 \simeq_B \alpha_2$ *if* $\sigma_1 \simeq_B \sigma_2[V]$*, i.e.* $\alpha_1 \preceq_B \alpha_2$ *and* $\alpha_2 \preceq_B \alpha_1$*.*

The relation $\alpha_1 \simeq_B \alpha_2$ *between narrowing steps defines a set of equivalence classes of narrowing steps. In what follows, we will be interested in choosing a unique representative* $\underline{\alpha} \in [\alpha]_{\simeq_B}$ *in each equivalence class of narrowing steps from* t*. Therefore,* $\underline{\alpha}$ *will always denote the chosen unique representative* $\underline{\alpha} \in [\alpha]_{\simeq_B}$ *that is minimal w.r.t. the order* $\preceq_B$*.*

The relation $\preceq_B$ provides an improvement on narrowing executions in two ways. First, narrowing steps with more general computed substitutions will be selected instead of narrowing steps with more specific computed substitutions. As a particular case, when both a rewriting step and a narrowing step are available for (even different positions of) the same term, the rewriting step will always be chosen. Second, the relation $\simeq_B$ provides a further optimization, since just one narrowing (or rewriting) step is chosen for each equivalence class, which further reduces the width of the narrowing tree.

The described strategy is formalized by the notion of *variant narrowing*.

Definition 7 (Variant Narrowing [36]**).** *Given a decomposition* $(\Sigma, B, \vec{E})$ *and a narrowing step* $\alpha : t \leadsto_{\sigma,\vec{E},B} t'$*,* α *is a* variant narrowing step *if it satisfies: (i)* $\sigma_{|Var(t)}$ *is* $(\vec{E}, B)$*-irreducible and (ii)* $\underline{\alpha}$ *is the chosen unique representative of its* $\simeq_B$*-equivalence class.*

Following the notation of [36], a variant narrowing step from t *to* t' *in* $(\Sigma, B, \vec{E})$ *with substitution* σ *is denoted as* $t \leadsto_{\sigma,\underline{\vec{E}},B} t'$*.*

A variant narrowing strategy could be easily defined by allowing only variant narrowing steps. However, we provide a more sophisticated narrowing strategy in the following section.

2.5. The folding variant narrowing strategy

The variant narrowing strategy defined above is a strategy in the sense of Definition 3, i.e., it always returns a subset of the narrowing steps that are available for each term. Note, however, that it has no memory of previous steps –just the input term to be narrowed– hence, it incurs no memory overhead. More sophisticated strategies can be developed by introducing some sort of memory that can avoid the repeated generation of useless or unnecessary computation steps. This is the case of the folding narrowing strategy of [36], which, when combined with the variant

narrowing strategy, provides the *folding variant narrowing strategy* which is complete for variant generation of a term and it terminates when the input term has a finite set of *most general variants.*

In Definition 8 below, we introduce a *folding narrowing* relation on term variants. Folding narrowing allows the deployed variant narrowing tree to be seen as a graph, where some leaves are connected to other nodes by implicit "fold" arrows. This definition normalizes each computed variant, which is not performed in the original definition of [36]. Note that we easily extend the variant narrowing strategy to variants, i.e., $(t,\theta) \rightsquigarrow_{\sigma,\vec{E},B} (t',\theta')$ iff $t \rightsquigarrow_{\sigma,\vec{E},B} t'$ and $\theta' = \theta\sigma$.

Definition 8 (Folding Variant Narrowing Strategy). *Let $\mathscr{R} = (\Sigma, B, \vec{E})$ be a decomposition. Given a Σ-term t, the frontier from term variant $I = (t, id)$ is defined as*

$$\begin{array}{l} \mathit{Frontier}(I)_0 = \{(t{\downarrow_{\vec{E},B}}, id)\}, \\ \mathit{Frontier}(I)_{n+1} = \{(y{\downarrow_{\vec{E},B}}, (\rho\sigma){\downarrow_{\vec{E},B}}) \mid (\exists (z,\rho) \in \mathit{Frontier}(I)_n : (z,\rho) \rightsquigarrow_{\sigma,\vec{E},B} (y,\rho\sigma)) \wedge \\ \qquad\qquad\qquad\qquad\qquad\qquad (\nexists k \leq n, (w,\tau) \in \mathit{Frontier}(I)_k : (w,\tau) \leq_{\vec{E},B} (y,\rho\sigma))\}, \\ \qquad\qquad\qquad\qquad\qquad\qquad n \geq 0 \end{array}$$

The folding variant narrowing strategy, denoted by $VN^{\circlearrowleft}_{\mathscr{R}}$, is defined as

$$VN^{\circlearrowleft}_{\mathscr{R}}(t) = \{\alpha \mid \alpha : t \rightsquigarrow^{k}_{\sigma,\vec{E},B} t' \wedge \exists k \geq 0 : (t',\sigma) \in \mathit{Frontier}((t,id))_k\}$$

Example 8. *For the input term $X \oplus Y$, the computed $VN^{\circlearrowleft}_{\mathscr{R}}$ steps in the equational theory of Example 2 are as follows.*

(i) $(X \oplus Y, id) \rightsquigarrow_{\phi_1} (X', \phi_1)$, *using* $\phi_1 = \{X \mapsto 0, Y \mapsto X'\}$ *and Equation* (1)*,*
(ii) $(X \oplus Y, id) \rightsquigarrow_{\phi_2} (X', \phi_2)$, *using* $\phi_2 = \{X \mapsto X', Y \mapsto 0\}$ *and Equation* (1)*,*
(iii) $(X \oplus Y, id) \rightsquigarrow_{\phi_3} (0, \phi_3)$, *using* $\phi_3 = \{X \mapsto X', Y \mapsto X'\}$ *and Equation* (2)*,*
(iv) $(X \oplus Y, id) \rightsquigarrow_{\phi_4} (Y', \phi_4)$, *using* $\phi_4 = \{X \mapsto Y' \oplus X', Y \mapsto X'\}$ *and Equation* (3)*,*
(v) $(X \oplus Y, id) \rightsquigarrow_{\phi_5} (Y', \phi_5)$, *using* $\phi_5 = \{X \mapsto X', Y \mapsto Y' \oplus X'\}$ *and Equation* (3)*,*
(vi) $(X \oplus Y, id) \rightsquigarrow_{\phi_6} (U \oplus Y', \phi_6)$, *using* $\phi_6 = \{X \mapsto X' \oplus U, Y \mapsto X' \oplus Y'\}$ *and Equation* (3)*.*

where all computed substitutions are normalized. Non-normalized narrowing steps such as

$(X \oplus Y, id) \rightsquigarrow_{\phi_9} (Y', \phi_9)$, *using* $\phi_9 = \{X \mapsto X' \oplus X', Y \mapsto Y'\}$ *and Equation* (3)

are not in $VN^{\circlearrowleft}_{\mathscr{R}}$ *because they are all subsumed by a variant narrowing step that computes the normalized version of the same substitution, e.g.,* $(Z,\phi_1) \leq_{\vec{E},B} (Z,\phi_9)$. *Furthermore, the sequence* $(X \oplus Y, id) \leadsto_{\phi_6} (Z_1 \oplus Z_2, \phi_6) \leadsto_{\phi'_6} (Z'_1 \oplus Z'_2, \phi_6\phi'_6)$ *corresponding to the two-step prefix of the infinite variant narrowing derivation of Example 6 is not in* $VN^{\circlearrowleft}_{\mathscr{R}}$ *because* $(Z_1 \oplus Z_2, \phi_6) \leq_{\vec{E},B} (Z'_1 \oplus Z'_2, \phi_6\phi'_6)$.

For a decomposition $(\Sigma, B, \vec{E})$, completeness of folding variant narrowing w.r.t. $(\vec{E},B)$-normalized substitutions is proved in [36, Theorem 4].

3. Specializing Convergent Equational Theories modulo Axioms

In this section, we introduce a partial evaluation algorithm for the decomposition $(\Sigma, B, \vec{E})$ of an equational theory $(\Sigma, \mathscr{E})$, with $\mathscr{E} = E \uplus B$, that is based on computing folding variant narrowing trees, and we establish the correctness of the transformation system. Our partial evaluation algorithm extends the general NPE procedure of [14], which is parametric w.r.t. an unfolding rule to construct finite derivations for an expression and an abstraction operator used to guarantee that only finitely many expressions are evaluated.

This section is organized as follows. In Section 3.1, we recall the key ideas of the NPE approach. In Section 3.2, we discuss how the specialization of programs that contain sorts, subsorts, rules, and equational axioms is significantly more elaborate. In Section 3.3, we present the general algorithm for order sorted equational partial evaluation modulo axioms based on folding variant narrowing. Local termination of the general algorithm is discussed in Section 3.4, whereas global termination is discussed in Section 3.5. In Section 3.6, a post-processing algorithm is presented that gets rid of unnecessary symbols and further optimizes the program.

3.1. The NPE Approach

Given a convergent set E of equations (oriented into a set $\vec{E}$ of rewrite rules) and a set Q of input terms, the aim of NPE [14] is to derive a new set $\vec{E}'$ of rules (called a partial evaluation of $\vec{E}$ w.r.t. Q, or a partial evaluation of Q in $\vec{E}$) which computes the same answers and irreducible forms (w.r.t. narrowing) as $\vec{E}$ for any term that is inductively covered (*closed*) by the calls in Q (henceforth called *specialized calls*). This is achieved by ensuring that every term (and subterm) in the leaves of the (partially unfolded) execution tree for each $t \in Q$ that can be narrowed in $\vec{E}$ can also be (correspondingly) narrowed in $\vec{E}'$. Roughly speaking, $\vec{E}'$ is obtained by first constructing a (possibly partial) *finite* narrowing tree for each input term t and then gathering together the set of *resultants* $t\theta_1 \Rightarrow t_1, \ldots, t\theta_k \Rightarrow t_k$ that can be constructed by considering the leaves of the tree, say $t_1, \ldots, t_k$, and the computed

substitutions $\theta_1, \ldots, \theta_k$ of the associated branches of the tree (i.e., a resultant rule is associated to each root-to-leaf derivation of the partial narrowing tree). Resultants perform what in fact is an n-step computation in $\vec{E}$, with $n > 0$, by means of a single step computation in $\vec{E}'$. The unfolding process is iteratively repeated for every narrowable subterm of $t_1, \ldots, t_k$ that is not covered by the root nodes of the already deployed narrowing trees. This ensures that resultants form a complete description covering all calls that may occur at run-time in $\vec{E}'$. As discussed below, local and global termination criteria are generally imposed to ensure both termination of the expansion of each narrowing tree and termination of recursively constructing narrowing trees, respectively.

In contrast to the partial deduction of logic programs, in classical NPE the notion of closedness is not a mere syntactic subsumption check that every subterm occurring in the leaves of the tree(s) is a substitution instance of one of the terms in Q. Instead, in order to properly deal with nested function calls, the closedness notion recurses over the structure of the terms [14]. Informally, a Σ-term t is considered Q-closed w.r.t. Σ (we often say that t is closed w.r.t. Q and Σ, or just closed w.r.t. Q when no confusion can arise) iff either: (i) it does not contain defined function symbols of Σ, or (ii) there exists a substitution θ and a (possibly renamed) $q \in Q$ such that $t = q\theta$, and the terms in θ are recursively Q-closed w.r.t. Σ. For instance, given a free binary constructor symbol $\bullet$ (i.e., it does not obey any structural axioms such as associativity or commutativity), the term $t = a \bullet (Z \bullet a)$ is closed w.r.t. $Q = \{a \bullet X, Y \bullet a\}$ or $\{X \bullet Y\}$, but it is not with Q being $\{a \bullet X\}$.

Let us illustrate the classical NPE method with the following example that shows its ability to perform *deforestation* [64], a popular transformation that neither standard partial evaluation nor partial deduction can achieve [14]. Essentially, the aim of deforestation is to eliminate useless intermediate data structures, thus reducing the number of passes over data.

Example 9. *Consider the following Maude program that computes the mirror image of a (non-empty) binary tree, which is built with the free constructor* `_{_}_` *that stores an element as root above two given (sub-)trees, its left and right children. Note that this is a simple specialization problem, where the considered program does not contain any equational attributes either for* `_{_}_` *or for the operation* `flip` *defined therein:*

```
fmod FLIP-TREE is protecting NAT .
 sort NatTree . subsort Nat < NatTree .
 vars R L : NatTree . var N : Nat .
 op _{_}_ : NatTree Nat NatTree -> NatTree .
 op flip : NatTree -> NatTree .
 eq flip(N) = N [variant] .
```

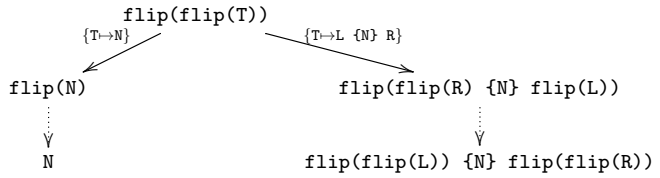

Figure 1: Folding variant narrowing tree for the goal `flip(flip(T))`.

```
 eq flip(L {N} R) = flip(R) {N} flip(L) [variant] .
endfm
```

By executing in Maude the input term `flip(flip(T))`*, this program returns the original tree* `T` *back, but it first computes an intermediate, mirrored tree* `flip(T)` *of* `T`*, which is then flipped again.*

Let us partially evaluate the input term `flip(flip(T))` *following the NPE approach. We compute the folding variant narrowing tree depicted*[10] *in Figure 1. This tree does not contain, altogether, uncovered calls in its leaves. Thus, after introducing the new symbol* `dflip`*, we get the following residual program:*

```
eq dflip(N) = N .
eq dflip(L {N} R) = dflip(L) {N} dflip(R) .
```

which is completely deforested, since the intermediate tree constructed after the first application of `flip` *is not constructed in the residual program using the specialized definition of* `dflip`*. This is equivalent to the program generated by deforestation [64] but with a much better*[11] *performance (see Section 5). Note that the fact that folding variant narrowing [36] ensures normalization of terms at each step is essential for computing the calls* `flip(flip(R))` *and* `flip(flip(L))` *that appear in the rightmost leaf of the tree in Figure 1, which are closed w.r.t. the root node of the tree.*

When we specialize programs that contain sorts, subsorts, rules, and equational axioms, things get considerably more involved, as discussed in the following section.

3.2. Partial evaluation of convergent rules modulo axioms

Let us motivate the problem by considering the following variant of the `flip` function of Example 9 for (binary) graphs instead of trees.

[10] We show narrowing steps in solid arrows and rewriting steps in dotted arrows.

[11] Similarly to [64], the optimal program `dflip(T) = T` cannot be produced by our equational NPE technique.

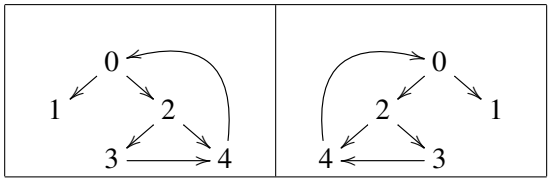

Figure 2: A binary graph (left) and its flipped version (right).

Example 10. *Consider the following Maude program for flipping binary graphs whose nodes may contain explicit, left and right references (pointers) to their child nodes in the graph. We use symbol* `#` *to denote an empty pointer. The* `BinGraph` *constructor* `_;_` *obeys associativity, commutativity, and identity (ACU) axioms so that it can be seen as a multiset of nodes* `{ R1 I R2 }`*, with* `R1` *and* `R2` *being references and* `I` *the node identifier. We provide for an unbounded number of (natural) node identifiers by establishing the subsort relation* `Nat < Id` *.*

```
fmod GRAPH is
  pr NAT .
  sorts BinGraph Node Id Ref .
  subsort Node < BinGraph . subsort Id < Ref . subsort Nat < Id .
  op {___} : Ref Id Ref -> Node . op mt : -> BinGraph .
  op _;_ : BinGraph BinGraph -> BinGraph [assoc comm id: mt] .
  op # : -> Ref . --- Void pointer
endfm
```

We are interested in flipping a graph and define a function `flip` *that takes a binary graph and returns the flipped graph.*

```
  var I : Id . vars R1 R2 : Ref . var BG : BinGraph .
  op flip : BinGraph -> BinGraph .
  eq [E1] : flip(mt) = mt  [variant] .
  eq [E2] : flip({R1 I R2} ; BG) = {R2 I R1} ; flip(BG) [variant] .
```

We can represent the graph shown on the left-hand side of Figure 2 as the following term `g` *of sort* `BinGraph`*:*

```
  { 1 0 2 } ; { # 1 # } ; { 3 2 4 } ; { # 3 4 } ; { # 4 0 }
```

By invoking `flip(g)`*, the graph shown on the right-hand side of Figure 2 is computed.*

In order to specialize the previous program for the call `flip(flip(BG))`, we need several PE ingredients that have to be generalized to the corresponding (order-sorted) *equational* notions: (i) *equational closedness*, (ii) *equational embedding*,

and (iii) *equational generalization*. These notions are discussed in the following sections.

3.3. Equational closedness and the generalized Partial Evaluation scheme

In order to extend the NPE approach to convergent equational theories $(\Sigma, B, \vec{E})$, we need to start by constructing a finite (possibly partial) $(\vec{E}, B)$-narrowing tree for each input term t in the set Q of specialized calls using the folding variant narrowing strategy [36], and then extracting the specialized rules $t\sigma \Rightarrow r$ (resultants) for each narrowing derivation $t \leadsto^{+}_{\sigma,\vec{E},B} r$ in the tree. In order to guarantee that all possible executions for t in the original program $(\Sigma, B, \vec{E})$ are covered by the specialization, we need to formalize an extended notion of closedness ensuring that any $(\vec{E}, B)$-narrowable subterm in the leaves of the tree can also be narrowed modulo B using the specialized rules. This ensures that resultants form a complete description covering all calls that may occur at run-time.

Let us define a general notion of *equational closedness* which relies on subsumption modulo B for theories whose function symbols (both defined and constructor symbols) can obey a set B of equational axioms.

Definition 9 (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 $\mathscr{C}$ of constructor symbols, so that $\Sigma = \mathscr{D} \uplus \mathscr{C}$. We say that a Σ-term t is closed* modulo *B (w.r.t. Q and Σ), or simply B-closed, if $closed_B(Q,t)$ holds, where the predicate $closed_B$ is defined as follows:*

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

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 also is closed modulo B. We often omit Σ when no confusion can arise.

Example 11. *In order to partially evaluate the program in Example 10 w.r.t. the input term* `flip(flip(BG))`*, we set $Q = \{$*`flip(flip(BG))`*$\}$ and start by con-*

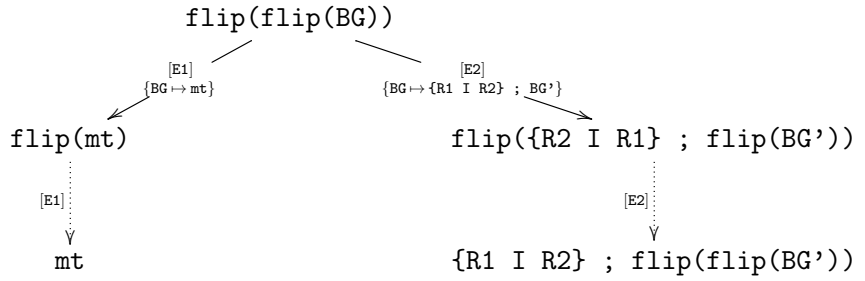

Figure 3: Folding variant narrowing tree for the goal `flip(flip(BG))`.

structing the folding variant narrowing tree that is shown[12] *in Figure 3.*

When we consider the leaves of the tree, we identify two requirements for Q-closedness, with B being ACU: (i) $closed_B(Q,t_1)$ with $t_1 =$ `mt` *and (ii) $closed_B(Q,t_2)$ with $t_2 =$* `{R1 I R2} ; flip(flip(BG'))`. *The call $closed_B(Q,t_1)$ holds straightforwardly (i.e., it is reduced to true) since the* `mt` *leaf is a constant and cannot be narrowed. The call $closed_B(Q,t_2)$ also returns true because* `_;_` *is constructor,* `{R1 I R2}` *is a flat constructor term, and* `flip(flip(BG'))` *is a (syntactic) renaming of the root of the tree.*

We now show an example that requires using B-matching in order to ensure equational closedness modulo B.

Example 12. *Let us add a new sort* `BinGraph?` *to the program in Example 10 to encode bogus graphs that may contain spurious nodes in a supersort* `Id?` *and homomorphically extend the rest of symbols and sorts. For simplicity, we just consider one additional constant symbol* `e` *of sort* `Id?`.

```
sorts BinGraph? Id? Node? Ref? .
subsorts BinGraph Node? < BinGraph? .
subsort Node < Node? .
subsort Id < Id? .
subsorts Ref Id? < Ref? .
op e : -> Id? .
op {___} : Ref? Id? Ref? -> Node? .
op _;_ : BinGraph? BinGraph? -> BinGraph? [assoc comm id: mt] .
vars I I1 : Id . var I? : Id? .
vars R1 R2 : Ref . vars R1? R2? : Ref? .
var BG : BinGraph . var BG? : BinGraph? .
```

[12]To ease reading, the arcs of the narrowing tree are labelled with the corresponding equation applied at each narrowing step.

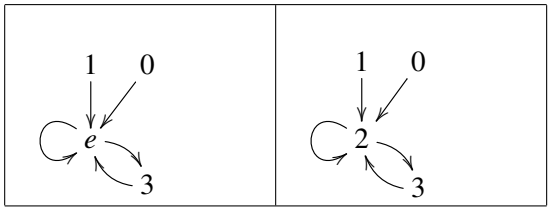

Figure 4: Fixing a graph.

Let us consider a function `fix` *that receives an extended graph* `BG?`*, an unwanted node* `I?`*, and a new content* `I`*, and that traverses the graph replacing* `I?` *by* `I`*.*

```
op fix : Id Id? BinGraph? -> BinGraph? .
eq [E3] : fix(I, I?, {R1? I? R2?} ; BG?) =
          fix(I, I?, {R1? I R2?} ; BG?) [variant] .
eq [E4] : fix(I, I?, {I? I1 R2?} ; BG?) =
          fix(I, I?, {I I1 R2?} ; BG?) [variant] .
eq [E5] : fix(I, I?, {R1? I1 I?} ; BG?) =
          fix(I, I?, {R1? I1 I} ; BG?) [variant] .
eq [E6] : fix(I, I?, BG) = BG [variant] .
```

For example, consider the following term `t` *of sort* `BinGraph?`*:*

```
{# 1 e} ; {e 0 #} ; {e e 3} ; {e 3 #}
```

that represents the graph shown on the left-hand side of Figure 4. By invoking `fix(2, e, t)`*, we can fix the graph* `t` *by computing the corresponding transformed graph shown on the right-hand side of Figure 4, where the unwanted constant* `e` *has been replaced with* 2*.*

Now assume we want to specialize the above function `fix` *w.r.t. the input term* `fix(2, e, {R1 I R2} ; BG?)`*, that is, a bogus graph with at least one non-spurious node* `{R1 I R2}` *(it is non-spurious because of the sort of variable* `I`*). Following the proposed methodology, we set* $Q =$ `{fix(2, e, {R1 I R2} ; BG?)}` *and start by constructing the folding variant narrowing tree shown in Figure 5.*

The right leaf `{R1 I R2} ; BG` *is a constructor term and cannot be unfolded. The first three branches to the left of the tree are closed modulo ACU with the root of the tree in Figure 5. For instance, for the left leaf*

$$t =$$ `fix(2, e, {R1?' 2 R2?'} ; BG?' ; {R1 I R2})`

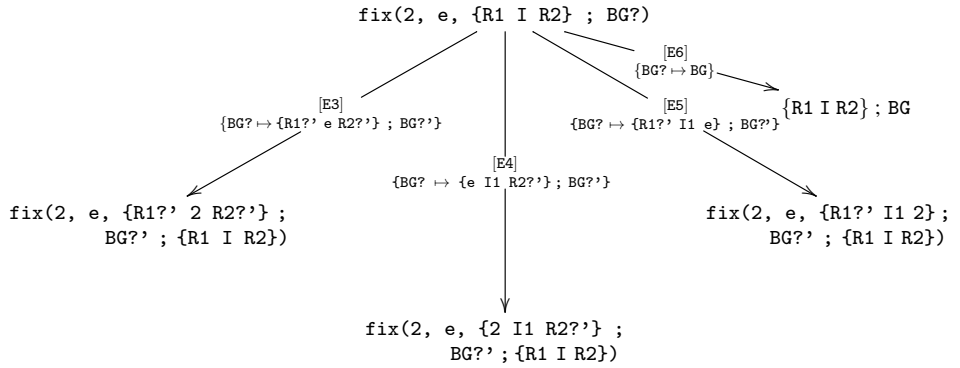

Figure 5: Folding variant narrowing tree for the goal `fix(2, e, {R1 I R2} ; BG?)`.

the condition $closed_B(Q,t)$ *is reduced*[13] *to true because* t *is an instance (modulo ACU) of the root node of the tree, and the subterm* $t' =$`({R1?' 2 R2?'} ; BG?')` *occurring in the corresponding ACU-matcher is a constructor term. The other two branches can be proved ACU-closed with the tree root in a similar way.*

Example 13 (Example 12 continued). *Now let us assume that the function* `flip` *of Example 10 is replaced by the following definition extended to (bogus graphs of sort)* `BinGraph?`*, where the former equation* `E2` *is an instance of the new equation* `E2x`*:*

```
op flip : BinGraph? -> BinGraph? .
eq [E1x] : flip(mt) = mt  [variant] .
eq [E2x] : flip({R1? I? R2?} ; BG?)
         = {R2? I? R1?} ; flip(BG?) [variant] .
```

We specialize the whole program containing functions `flip` *and* `fix` *w.r.t. input term* `flip(fix(2, e, flip(BG)))`*; that is, take a graph* `BG`*, flip it, then fix any occurrence of nodes* `e`*, and finally flip it again. A partial folding variant narrowing tree for the input term is shown in Figure 6. Note that the right branch of the tree has been stopped at this point (the stopping criterion is formalized in Section 3.4 below) to avoid infinite unfolding by using equation* `E2x`*. Unfortunately this tree does not represent all possible computations for (any ACU-instances of) the input term, since the narrowable redexes occurring in the tree leaves are not*

[13]Note that this is only true because pattern matching modulo *ACU* is used for checking closedness.

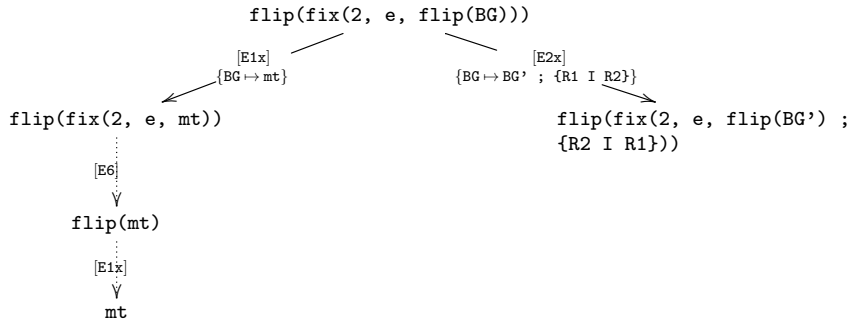

Figure 6: Folding variant narrowing tree for the goal `flip(fix(2, e, flip(BG)))`.

a recursive instance of the only partially evaluated call so far, `flip(fix(2, e, flip(BG)))`. *That is, the term*

```
flip(fix(2, e, flip(BG') ; {R2 I R1}))
```

in the rightmost leaf is not ACU-closed w.r.t. the root node of the tree. As in NPE, we need to introduce a methodology that recurses (modulo B) over the structure of the terms to augment the set of specialized calls in a controlled way, so as to ensure that all possible calls are covered by the specialized program.

We are now ready to formulate the backbone of our partial evaluation methodology for equational theories that crystallize the ideas of the example above. We define a generic algorithm (Algorithm 1) that is parameterized by:

1. a *narrowing relation* (with narrowing strategy $\mathscr{S}$) that constructs search trees,
2. an *unfolding rule* UNFOLD that determines when and how to terminate the construction of the trees, and
3. an *abstraction operator* ABSTRACT 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.

Note that, by using the notion of decomposition, the partial evaluation of an equational theory $(\Sigma, E \uplus B)$ can be seen as a particular case of this parameterized algorithm that is defined for rewrite theories $(\Sigma, B, \vec{E})$, and we prefer to keep it generic for further instantiations of this algorithm to deal with more complex rewrite theories.

Informally, the algorithm proceeds as follows. Given the input theory $\mathscr{R}$ and the set of terms Q, the first step consists in applying the unfolding rule UNFOLD(Q,$\mathscr{R}$,$\mathscr{S}$) to compute a finite (possibly partial) narrowing tree in $\mathscr{R}$ for

Algorithm 1 Partial Evaluation for Equational Theories

Require:

An order-sorted rewrite theory $\mathscr{R} = (\Sigma, B, \vec{E})$ and a set of terms Q to be specialized in $\mathscr{R}$, and a narrowing strategy $\mathscr{S}$

Ensure:

A set Q' of terms s.t. Q and $\text{UNFOLD}(Q', \mathscr{R}, \mathscr{S})$ are closed modulo B w.r.t. Q'

1: **function** EQNPE($\mathscr{R}$,Q,$\mathscr{S}$)
2: $Q \leftarrow Q{\downarrow_{\vec{E},B}}$
3: **repeat**
4: $Q' \leftarrow Q$
5: $\mathscr{L} \leftarrow \text{UNFOLD}(Q', \mathscr{R}, \mathscr{S})$
6: $Q \leftarrow \text{ABSTRACT}(Q', \mathscr{L}, B)$
7: **until** $Q' =_B Q$
8: **return** Q'

each term t in Q and return the set $\mathscr{L}$ of the (normalized) leaves of the trees. Then, instead of proceeding directly with the partial evaluation of the terms in $\mathscr{L}$, an abstraction operator $\text{ABSTRACT}(Q, \mathscr{L}, B)$ is applied that properly combines each uncovered term in $\mathscr{L}$ with the (already partially evaluated) terms of Q so that the infinite growing of Q is avoided. The abstraction phase yields a new set of terms which may need further specialization, and, thus, the process is iteratively repeated while new terms are introduced.

Note that Algorithm 1 does not explicitly compute a partially evaluated theory $\mathscr{R}' = (\Sigma, B, \vec{E}')$. It does so implicitly, by computing the set Q' of partially evaluated terms (that unambiguously determine $\vec{E}'$ as the set $R_{Q',\mathscr{R}}$ of resultants $t\sigma \Rightarrow r$ associated to the root-to-leaf derivations $t \leadsto^{+}_{\sigma,\vec{E},B} r$ in the tree, with t in Q'), such that the closedness condition for $\vec{E}'$ modulo B w.r.t. Q' is satisfied.

For the correctness of Algorithm 1, we require any instance of the generic abstraction operator $\text{ABSTRACT}(Q, \mathscr{L}, B)$ to agree with the following definition.

Definition 10 (Equational Abstraction). *Given the finite set of terms T and the already evaluated set of terms Q, $\text{ABSTRACT}(Q, T, B)$ 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, and*
2. *for all $t \in (Q \cup T)$, t is closed with respect to Q' modulo B.*

Roughly speaking, condition (1) ensures that the abstraction operator does not "create" new function symbols (i.e., symbols not present in the input arguments),

whereas condition (2) ensures that the resulting set of terms "covers" (modulo B) the calls previously specialized and that equational closedness is preserved throughout successive abstractions.

There are two correctness issues for a PE procedure: *termination*, i.e., given any input goal, execution should always reach a stage at which there is no way to continue; and (partial) *correctness and completeness*, i.e., the residual program behaves as the original one for the considered input terms (provided PE terminates). The basic completeness of the transformation is ensured whenever $\mathscr{R}'$ is closed modulo B w.r.t. Q', i.e., every call in (the right-hand side of the rules in) $\mathscr{R}'$ is a (recursive) instance (modulo B) of a term in Q'.

The following lemma is the main result in this section and establishes that the function EQNPE of Algorithm 1 reaches the B-closedness condition upon termination, independently from the narrowing strategy, unfolding rule, and abstraction operator. This is a key property of partial evaluation frameworks that is necessary to achieve completeness of the specialization.

Lemma 11. *Let $\mathscr{R} = (\Sigma, B, \vec{E})$ be a decomposition of an equational theory $(\Sigma, \mathscr{E})$, $\mathscr{S}$ a narrowing strategy, and Q a set of terms. If* EQNPE$(\mathscr{R}, Q, \mathscr{S})$ *terminates computing the set Q' of terms, then: (1) Q is B-closed w.r.t. Q', and (2) also the rules in the resulting partially evaluated theory $\mathscr{R}'$ are B-closed w.r.t. Q'.*

In order to ensure the termination of the algorithm, the partial narrowing trees must be finite and the iterative construction of the partial trees must eventually terminate while still guaranteeing that the desired amount of specialization is retained and that the equational closedness condition is reached. In the following subsection, we present a simple but useful solution to the termination problem by introducing appropriate unfolding and abstraction functions that fit the narrowing strategies described in Section 2 for specializing equational theories.

3.4. Termination of the PE process

Partial evaluation involves two classical termination problems: the so-called *local* termination problem (the termination of unfolding, or how to control and keep the expansion of the narrowing trees finite, which is managed by an unfolding rule), and *global* termination (which concerns termination of recursive unfolding, or how to stop recursively constructing more and more narrowing trees).

The problem of obtaining (sensibly expanded) finite narrowing trees essentially boils down to defining suitable unfolding rules that somehow ensure that infinite unfolding is not performed. In the following section, we introduce an unfolding rule that attempts to maximize unfolding while retaining termination. Our strategy is based on the use of a homeomorphic embedding relation (i.e., a

structural preorder) under which a term t is greater than (i.e., it embeds) another term t', written as $t' \trianglelefteq t$, if t' can be obtained from t by deleting some parts (e.g., $s(0+s(X))*s(X+Y)$ embeds $s(X)*s(Y)$), which we suitably define to work modulo axioms B. Homomorphic embedding modulo B is used by our strategy as a way to stop (R,B)-narrowing derivations.

Embedding relations are very popular for ensuring termination of symbolic methods and program optimization techniques because embedding relations are well-quasi-orderings, i.e., given a finite signature, for every infinite sequence of terms $t_1, t_2, \ldots$, there exist $i < j$ such that $t_i \trianglelefteq t_j$. 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 [47]: whenever a new expression t_{n+1} is to be added to the sequence, we first check whether t_{n+1} embeds any of the expressions already in the sequence. If that is the case, we say that $\trianglelefteq$ whistles, i.e., it has detected (potential) non-termination and the computation has to be stopped. Otherwise, t_{n+1} can be safely added to the sequence and the computation proceeds. For instance, if we work modulo commutativity, we must stop a sequence where the term $s(0+s(X))*s(X+Y)$ occurs after $s(Y)*s(X+0)$ since it embeds it modulo the commutativity of $+$ and $*$. This is simple but less crude than imposing ad hoc depth bounds and it still guarantees termination (finite unfolding) in all cases.

Following [6, 7], we extend the homeomorphic embedding ("syntactically simpler") relation on nonground terms [47] to the order-sorted, *semantic* case of working modulo axioms. Variations of [47] are used in termination proofs for term-rewriting systems [30] and for ensuring local termination of partial deduction [17]. Given a term t, we let $\lceil t \rceil$ denote the kind of t, i.e., $\lceil t \rceil = [ls(t)]$.

Definition 12 (Order-sorted Symbolic Homeomorphic Embedding [7]**).** *The order-sorted symbolic embedding relation $\breve{\trianglelefteq}$ over $\mathscr{T}_\Sigma(\mathscr{X})_\mathsf{s}$ is defined as follows*

Variable	Diving	Coupling
$\dfrac{\lceil x \rceil = \lceil y \rceil}{x \breve{\trianglelefteq} y}$	$\dfrac{\exists i \in \{1,\ldots,n\} : s \breve{\trianglelefteq} t_i}{s \breve{\trianglelefteq} f(t_1,\ldots,t_n)}$	$\dfrac{\forall i \in \{1,\ldots,n\} : s_i \breve{\trianglelefteq} t_i}{f(s_1,\ldots,s_n) \breve{\trianglelefteq} f(t_1,\ldots,t_n)}$

Roughly speaking, the Variable inference rule allows dealing with (order-sorted) variables in terms, while the Diving and Coupling inference rules are similar to the pure (syntactic) homeomorphic embedding definition. Since we assume that Σ has no ad-hoc overloading, the terms $f(s_1,\ldots,s_n)$ and $f(t_1,\ldots,t_n)$ in the Coupling inference rule belong to the same kind; hence, no extra check is needed. Nor is an extra check required for the Diving rule for the same reason since it simply traverses the term structure.

It is worth noting that, while it seems natural to consider that $X{:}A \mathrel{\breve{\trianglelefteq}} Y{:}B$ for the case when A is a subsort of B (which holds because $[A] = [B]$, hence $\lceil x \rceil = \lceil y \rceil$), the practical usefulness of having $X{:}A \mathrel{\breve{\trianglelefteq}} Y{:}B$ is less evident for the case when B is a subsort of A (which holds for the very same reason). However, consider an overloaded operator g, with $g : A \to A$ and $g : B \to B$, with $A > B$. In a context of symbolic execution where logical variables are considered, it could be the case of having a computation sequence $(t_1, \ldots t_i, \ldots, t_j,\ t_{j+1} \ldots)$, with $t_i = g(X{:}A)$, $t_j = g(Y{:}B)$, and $t_{j+1} = g(X{:}A)$, where t_{j+1} derives from t_j by a symbolic execution step (e.g., think of a narrowing step from $g(Y{:}B)$ by using a program rule $g(g(X{:}A)) \to g(X{:}A)$). Hence, the fact that $g(X{:}A) \mathrel{\breve{\trianglelefteq}} g(X{:}B)$ allows any risk of non-termination to be detected at t_j, i.e., before generating t_{j+1}; this prevents an infinite sequence like $g(X{:}A), \ldots, g(X'{:}A), \ldots, g(X''{:}A), \ldots$ from being generated.

In [7], the equational extension (modulo B) of $\breve{\trianglelefteq}$, in symbols $\breve{\trianglelefteq}_B$, is defined in the natural way as follows.

Definition 13 (Order-sorted Symbolic Homeomorphic Embedding Modulo B [5]). *The order-sorted B-embedding relation $\breve{\trianglelefteq}_B$ is $(\overset{ren}{=}_B) \circ (\breve{\trianglelefteq}) \circ (\overset{ren}{=}_B)$, where $v \overset{ren}{=}_B v'$ iff there is a renaming substitution σ for v' such that $v =_B v'\sigma$.*

Example 14. *Consider the following equational theory (written in Maude syntax) that defines the signature of natural numbers. The defined sort hierarchy has top sort* `Nat` *and (disjoint) subsorts* `Zero` *and* `NzNat` *(for non-zero natural numbers). The sort* `Nat` *is generated from the constant* `0` *(of sort* `Zero`*) and the successor operator* `suc`[14] *(of sort* `NzNat`*). We also define the associative and commutative natural addition operator symbol* `_+_` *for sort* `Nat` *but add two extra subsort-overloaded definitions.*

```
fmod NAT is
  sorts Zero NzNat Nat .
  subsorts Zero NzNat < Nat .
  op 0 : -> Zero .
  op suc : Nat -> NzNat .
  op _+_ : Zero Zero -> Zero [assoc comm] .
  op _+_ : NzNat Nat -> NzNat [assoc comm] .
  op _+_ : Nat Nat -> Nat [assoc comm] .
endfm
```

Then, we have `1 + X:Nat` $\mathrel{\breve{\trianglelefteq}_B}$ `Y:Nat + (1 + 2)` *because* `Y:Nat + (1 + 2)` *is equal to* `1 + (2 + Y:Nat)` *modulo the associativity and commutativity of* `_+_`*,*

[14] For simplicity, we represent natural numbers in decimal notation; e.g., `2` for `suc(suc(0))`.

and `1 + X:Nat` *is homeomorphically embedded into* `1 + (2 + Y:Nat)`. *Similarly,* `1 + X:Zero` $\check{\trianglelefteq}_B$ `Y:Nat + (1 + 2)` *and* `1 + X:NzNat` $\check{\trianglelefteq}_B$ `Y:Nat + (1 + 2)` *hold.*

A *well-quasi ordering* $\preceq$ on terms is a transitive and reflexive binary relation such that, for any infinite sequence of terms $t_1, t_2, \ldots$ with a finite number of operators, there exist i, j with $i < j$ and $t_i \preceq t_j$. Similarly to the pure homeomorphic embedding $\trianglelefteq$, the order-sorted symbolic homeomorphic embedding relation $\check{\trianglelefteq}_B$ is a well-quasi-order on the set of terms $\mathscr{T}_\Sigma(\mathscr{X})$ for *class-finite*[15] theories.

Lemma 14. *Given a* class-finite *theory* (Σ, B)*, the order-sorted symbolic homeomorphic embedding relation* $\check{\trianglelefteq}_B$ *is a well-quasi ordering on the set* $\mathscr{T}_\Sigma(\mathscr{X})$.

Strictly speaking, the restriction to class-finite theories rules out theories that contain identity axioms. However, dealing with identity poses no practical problem for partial evaluation based on (folding) variant narrowing. This is because the deployed (folding) variant narrowing traces only contain $(\vec{E}, B)$-normalized terms; hence, embedding tests are always performed between terms in which the identity elements have been already removed by $(\vec{E}, B)$-normalization.

In this paper, we use the high-performance implementation of the order-sorted symbolic homeomorphic embedding relation $\check{\trianglelefteq}_B$ given in [7], where a comparison of increasingly efficient implementations of $\check{\trianglelefteq}_B$ can be found. State of the art local control rules based on homeomorphic embedding do not check for embedding against all previously selected expressions but rather only against those in its sequence of *covering ancestors* [19]. This increases the efficiency of the checking and allows the whistle $\check{\trianglelefteq}_B$ to blow later. The following unfolding function instantiates the function UNFOLD of Algorithm 1 by using the embedding relation in a constructive way to produce finite narrowing trees and then extract the leaves from the trees.

We need the following auxiliary notion. We say that a narrowing derivation D is *admissible w.r.t.* $\check{\trianglelefteq}_B$ 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 $s \check{\trianglelefteq}_B t$.

Definition 15 (Unfolding function). *Given the rewrite theory* $\mathscr{R} = (\Sigma, B, \vec{E})$ *and a term* t_0 *to be specialized in* $\mathscr{R}$*, we define* UNFOLD$(t_0, \mathscr{R}, \mathscr{S})$*, for* $\mathscr{S} = VN^{\circlearrowleft}_{\mathscr{R}}$*, as*

[15] B is called *class-finite* if all of the B-equivalence classes of terms in the quotient term algebra $\mathscr{T}_\Sigma(\mathscr{X})/=_B$ are finite. This includes the class of permutative equational theories B, where $s =_B t$ implies that the terms s and t contain the same symbols with the same number of occurrences [20]. The class of permutative equational theories includes any combination of associativity and/or commutativity axioms.

the set of terms

$$Unfold^{\breve{\trianglelefteq}_B}(t_0,\mathscr{R}) = \{t_n \mid t_0 \rightsquigarrow^n t_n \in VN^{\circlearrowleft}_{\mathscr{R}}(t_0),\\ t_0 \rightsquigarrow^{n-1} t_{n-1} \text{ is admissible w.r.t. } \breve{\trianglelefteq}_B \text{ and}\\ \text{either } \nexists w : t_0 \rightsquigarrow^n t_n \rightsquigarrow w \in VN^{\circlearrowleft}_{\mathscr{R}}(t_0)\\ \text{or } t_0 \rightsquigarrow^n t_n \text{ is not admissible w.r.t. } \breve{\trianglelefteq}_B.\}$$

Given a set Q of terms, we also define $Unfold^{\breve{\trianglelefteq}_B}(Q,\mathscr{R}) = \bigcup_{t\in Q} Unfold^{\breve{\trianglelefteq}_B}(t,\mathscr{R})$.

Note that the function $Unfold^{\breve{\trianglelefteq}_B}(Q,\mathscr{R})$ of Definition 15 computes a finite (possibly partial) folding variant narrowing tree in $\mathscr{R}$ 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 15 (Example 13 continued). *Consider again the (partial) folding variant narrowing tree of Figure 6. The narrowing redex*

$$t = \texttt{flip(fix(2, e, flip(BG')) ; \{R2 I R1\})}$$

in the right branch of the tree embeds modulo ACU the tree root

$$u = \texttt{flip(fix(2, e, flip(BG)))}$$

Since the whistle $u \breve{\trianglelefteq}_B t$ blows, the unfolding of this branch is stopped.

The following result establishes the termination of the unfolding process.

Theorem 1 (Local Termination). *Let $\mathscr{R} = (\Sigma, B, \vec{E})$ be a decomposition of an equational theory $(\Sigma, E \uplus B)$ and Q be a finite set of terms. The computation of $Unfold^{\breve{\trianglelefteq}_B}(Q,\mathscr{R})$ terminates.*

Nontermination of the function EQNPE in Algorithm 1 can be caused not only by the creation of an infinite narrowing tree but also by never reaching the equational closedness condition. Unlike local control, which is parametric w.r.t. the decision whether to stop or to proceed with the expansion, since it is safe to terminate the evaluation at any point, the global control does not allow this flexibility because we cannot stop the iterative extension of the set Q of partially evaluated expressions until all function calls in this set are B-closed w.r.t. Q itself.

3.5. Global Termination of Equational NPE

For global termination, partial evaluation relies on an abstraction operator to ensure that the iterative construction of a sequence of partial narrowing trees terminates while still guaranteeing that the desired amount of specialization is retained and that the equational closedness condition is reached. In order to avoid constructing infinite sets, instead of just taking the union of the set $\mathscr{L}$ of (possibly non-closed modulo B) terms in the leaves of the tree and the set Q of specialized calls, the sets Q and $\mathscr{L}$ are *generalized*. Hence, the abstraction operator returns a safe approximation A of $Q \cup \mathscr{L}$ so that each expression in the set $Q \cup \mathscr{L}$ is closed w.r.t. A. Let us show how we can define a suitable abstraction operator by using the notion of *equational least general generalization* modulo B (lgg_B) [8] so that we do not lose too much precision despite the abstraction. For more sophisticated global control, homeomorphic embedding can be combined with other techniques such as global trees, characteristic trees, or trace terms (see, e.g., [48] and its references).

Generalization is the dual of unification [59]: generalization (resp. unification) appear as the supremum (resp. infimum) operator in the lattice order of (unsorted) terms (up to renaming). Roughly speaking, the generalization problem (also known as *anti-unification*) for two or more expressions means finding their *least general generalization*, 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 `father(X,Y)` is a generalizer of both `father(john,sam)` and `father(tom,sam)`, but their least general generalizer is `father(X,sam)`.

For order-sorted theories, neither more general unifiers (mugs) nor least general generalizers (lggs) are generally unique, but there are always finite sets of them [9]. In [10], the notion of least general generalization is extended to work modulo equational axioms B, where function symbols can obey any combination of associativity, commutativity, and identity axioms (including the empty set of such axioms). Unlike the untyped case, there is in general no unique lgg in the framework of [8], due to both the order-sortedness and to the equational axioms. Instead, there is always a finite, minimal and complete set of lggs so that any other generalizer has at least one of them as a B-instance.

Formally, given an order-sorted signature Σ and a set of algebraic axioms B, a *generalization* modulo B of the nonempty set of Σ-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, $Dom(\theta_i) = Dom(\theta_j)$ for all $i,j \in \{1,\ldots,n\}$, and 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. In this paper, we use the high-performance implementation of the order-sorted least general generalization modulo B given in [4].

Given the current set Q of already specialized calls, in order to augment Q with a new set T of terms, the abstraction operator $\textsc{Abstract}(Q,T,B)$ of Algorithm 1 is particularized to an abstraction function that relies on the notion of *best matching set* (BMS), an order-sorted equational extension of [1] that is aimed at avoiding loss of specialization due to generalization. The notion of BMS is used in the abstraction process when selecting the most appropriate terms of Q to be selected for generalizing T, in the sense of providing least general generalizations.

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 present a simple motivating example, with $B = \emptyset$.

Example 16. *Let $Q = \{f(g(x)), f(g(a)), f(z)\}$ and $t = f(g(b))$. To compute the best matching set for t in Q, we first consider the set*

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

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

Definition 16 (Best Matching Set modulo B). *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 definition.

Example 17. *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 constructor 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:*

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

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 able to instantiate the function ABSTRACT(Q,T,B) of Algorithm 1 with the following equational abstraction function $abs^{\check{\trianglelefteq}_B}(Q,T)$ that relies on $\check{\trianglelefteq}_B$, the notion of best matching set, and equational least general generalization.

Definition 17 (Equational Least General Abstraction Function). *Let Q,T be two sets of terms. We define $abs^{\check{\trianglelefteq}_B}(Q,T)$ as follows:*

$$\begin{cases} abs^{\check{\trianglelefteq}_B}(\ldots abs^{\check{\trianglelefteq}_B}(Q,\{t_1\}),\ldots,\{t_n\}) & \text{if } T = \{t_1,\ldots,t_n\}, n > 1 \\ Q & \text{if } T = \emptyset \text{ or } T = \{X\}, \text{with } X \in \mathscr{X} \\ abs^{\check{\trianglelefteq}_B}(Q,\{t_1,\ldots,t_n\}) & \text{if } T = \{t\}, \text{with } t = c(t_1,\ldots,t_n),\ c \in \mathscr{C} \\ generalize_B(Q,Q',t) & \text{if } T = \{t\}, \text{with } t = f(t_1,\ldots,t_n),\ f \in \mathscr{D} \end{cases}$$

where $Q' = \{t' \in Q \mid root(t) = root(t') \text{ and } t' \check{\trianglelefteq}_B t\}$, and the function generalize *is:*

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

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

Roughly speaking, the equational least general abstraction function proceeds as follows. We distinguish the cases when the considered term t either: i) is a variable, or ii) is not a variable. In the first case, the term is simply ignored. In the second case, if t does not B-embed any term in Q, it is just added. However, if t B-embeds some *comparable* term in Q, we distinguish two cases. If t is already Q-closed, then it is simply discarded. Otherwise, the given term is generalized by computing the lgg_B of t w.r.t. each of its best matching terms, and the abstraction function is recursively applied to add the B-normalized version of w and of the terms in the matching substitutions θ_1 and θ_2.

Note that the roles of Q and T in the definition of $abs^{\check{\trianglelefteq}_B}(Q,T)$ are not symmetric, that is, $abs^{\check{\trianglelefteq}_B}(Q,T) \neq abs^{\check{\trianglelefteq}_B}(T,Q)$ in general. This is deliberate since when we add a new set T of calls to the set Q of already specialized calls, we want to preserve as much as possible the potential specialization achieved by partially evaluating Q. For instance, for $B = \emptyset$, $Q = \{f(g(1))\}$, and $T = \{g(1), f(1)\}$, we have $abs^{\check{\trianglelefteq}_B}(Q,T) = \{f(g(1)), g(1), f(1)\}$ whereas $abs^{\check{\trianglelefteq}_B}(T,Q) = \{g(1), f(X)\}$ because

$f(1) \check{\trianglelefteq}_B f(g(1))$ but $f(g(1)) \not\check{\trianglelefteq}_B f(1)$. Hence, $abs^{\check{\trianglelefteq}_B}(T,Q)$ does not achieve *polyvariant specialization*[16]. Actually, it would unnecessarily replace with $f(X)$ the original specialized call $f(g(1))$ (while simply adding $f(1)$ does not jeopardize termination as $f(g(1)) \not\check{\trianglelefteq}_B f(1)$), thus losing any potential specialization.

The following results establish the correctness and termination of the equational least general abstraction function.

Proposition 18. *The function $abs^{\check{\trianglelefteq}_B}$ of Definition 17 is an abstraction operator in the sense of Definition 10.*

Theorem 2. *The equational least general abstraction function $abs^{\check{\trianglelefteq}_B}$ terminates.*

Finally, the main result of this section follows from Theorem 1 and Proposition 18.

Theorem 3 (Global Termination). *Algorithm 1 terminates for the unfolding function $Unfold^{\check{\trianglelefteq}_B}$ and the equational least general abstraction function $abs^{\check{\trianglelefteq}_B}$.*

Let us illustrate the use of $abs^{\check{\trianglelefteq}_B}$ in the following specialization problem.

Example 18. *Let us consider again Example 17 and assume $Q = \{1 \otimes g(X)\}$ and $T = \{g(1) \otimes 1 \otimes g(Y)\}$. The call $abs^{\check{\trianglelefteq}_B}(Q,T)$ invokes*

$$generalize_B(Q, Q', g(1) \otimes 1 \otimes g(Y))$$

with $Q' = \{1 \otimes g(X)\}$, which in turn calls $abs^{\check{\trianglelefteq}_B}(Q \setminus BMS_B(Q',t), Q'')$, where

$$\begin{array}{ll} BMS_B(Q',t) = & \{1 \otimes g(X)\} \\ Q'' = & \{Z \otimes 1, Z \otimes g(W), g(X), g(1) \otimes g(Y), 1 \otimes g(Y), 1 \otimes g(X)\} \end{array}$$

This in turn calls to $abs^{\check{\trianglelefteq}_B}(\emptyset, Q'')$, which amounts to the sequence of imbricated calls

$$abs^{\check{\trianglelefteq}_B}(abs^{\check{\trianglelefteq}_B}(\emptyset, Z \otimes 1), \{Z \otimes g(W), g(X), g(1) \otimes g(Y), 1 \otimes g(y), 1 + g(X)\})$$

and since terms $Z \otimes 1, Z \otimes g(W)$ and $g(X)$ do not embed $1 \otimes g(X)$, then the three terms are added yielding the new call

$$abs^{\check{\trianglelefteq}_B}(\{Z \otimes 1, Z \otimes g(W), g(X)\}, \{g(1) \otimes g(Y), 1 \otimes g(y), 1 \otimes g(X)\})$$

that returns the set $\{Z \otimes 1, Z \otimes g(W), g(X)\}$ since all three terms $g(1) \otimes g(Y), 1 \otimes g(y)$ and $1 + g(X)$ are AC-closed. That is, $abs^{\check{\trianglelefteq}_B}(Q,T) = \{Z \otimes 1, Z \otimes g(W), g(X)\}$.

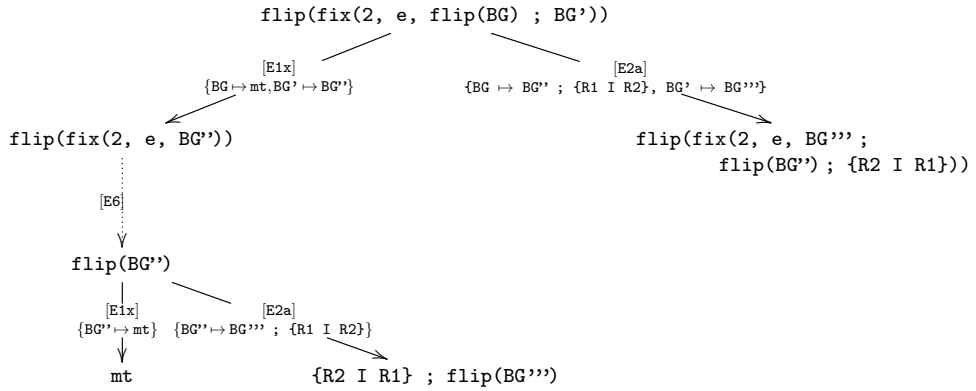

Figure 7: Folding variant narrowing tree for the goal `flip(fix(2, e, flip(BG) ; BG''))`.

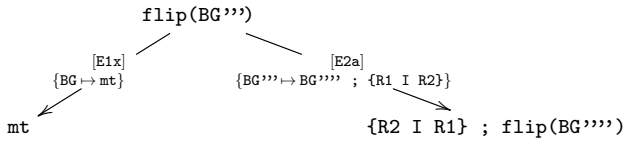

Figure 8: Folding variant narrowing tree for the goal `flip(BG''')`.

Example 19 (Example 15 continued). *Consider again the (partial) folding variant narrowing tree of Figure 6 with the leaf* $t =$ `flip(fix(2, e, flip(BG') ; {R2 I R1}))` *in the right branch of the tree and the tree root* $u =$ `flip(fix(2, e, flip(BG)))`. *We apply the equational least general abstraction function with* $Q = \{u\}$ *and* $T = \{t\}$.

Since t *is operation-rooted, we call* $generalize_B(Q, Q', t)$ *with* $Q' = Q$, *which in turn calls to* $abs^{\trianglelefteq_B}(Q \setminus BMS_{ACU}(Q', t), Q'')$, *with* $BMS_{ACU}(Q', t) = Q$ *and* $Q'' = \{w, v\}$, *where* $w =$ `flip(fix(2,e,flip(BG);BG'))` *is the only ACU least general generalization of* u *and* t *and* $v =$ `{R2' I' R1'}`. *Then the call returns the set* $\{w\}$. *However, this means that the previous folding narrowing tree of Figure 6 is now discarded, since the previous set of input terms* $Q = \{u\}$ *is now replaced by* $Q' = \{w\}$.

We start from scratch and the tree that results for the new call w *is shown in Figure 7. The right leaf embeds the root of the tree and is B-closed w.r.t. it. The left leaf* `mt` *is a constructor term. For the middle leaf* $t'' =$ `{R2 I R1} ; flip(BG''')` *the whistle* `flip(BG'')` $\trianglelefteq_{ACU}$ t'' *blows and we stop the derivation. However, it is*

[16]Polyvariance [51] is the ability to produce more than one specialized definitions for a single original function in the same specialization.

not B-closed w.r.t. w and we have to add it to the set Q', obtaining the new set of input terms $Q'' = \{w$,`flip(BG''')`$\}$. The specialization of the call `flip(BG''')` *amounts to constructing the folding variant narrowing tree of Figure 8, which is trivially ACU-closed w.r.t. its root.*

Example 20 (Example 19 continued). *Since the two trees in Figures 7 and 8 do represent all possible computations for (any ACU-instance of)* $u =$ `flip(fix(2, e, flip(BG)))`*, the partial evaluation process ends. Actually, u is an instance of the root of the tree in Figure 7 with* `{BG' ↦ mt}` *because of the identity axiom. The set* $Q''' = \{$`flip(fix(2,e,flip(BG);BG'))`$,$`flip(BG''')`$\}$ *is the computed specialization. Now we can extract the set of resultants $t\sigma \Rightarrow r$ associated to the root-to-leaf narrowing derivations in the two trees, which yields:*

```
eq flip(fix(2, e, flip(mt))) = mt .
eq flip(fix(2, e, flip({R1 I R2} ; BG'))) =
   flip(fix(2, e, flip(BG') ; {R2 I R1})) .
eq flip(fix(2, e, flip(mt) ; mt)) = mt .
eq flip(fix(2, e, flip(mt) ; BG ; {R1 I R2})) =
   {R2 I R1} ; flip(BG) .
eq flip(fix(2, e, flip({R1 I R2} ; BG) ; BG')) =
   flip(fix(2, e, flip(BG) ; {R2 I R1} ; BG')) .
eq flip(mt) = mt .
eq flip(BG ; {R1 I R2}) = {R2 I R1} ; flip(BG) .
```

The reader may have realized that the specialization call `flip(fix(2,e,flip(BG)))` should really return the same term `BG`, since the variable `BG` is of sort `BinGraph` instead of `BinGraph?`, i.e., `flip(fix(2, e, flip(BG))) = BG`. The resultants above traverse the given graph and return the same graph. Though the code may seem inefficient, we have considered this example because it allows all the different stages of the PE process to be illustrated.

The following example shows how a better specialization can be obtained.

Example 21. *Let us now overload the* `flip` *operator, having simultaneously two declarations for the* `flip` *symbol that are related in the subsort ordering* `Bingraph < Bingraph?`*:*

```
op flip : BinGraph -> BinGraph .
op flip : BinGraph? -> BinGraph? .
```

and four equations: `E1`*,* `E2`*,* `E2a`*, and* `E2b`*. By specializing the call* $t =$ `flip(fix(2,e,flip(BG)))`*, the subtype definition of* `flip` *allows Maude to simplify the term t using equation* `E6`*, which eliminates the occurrence of the*

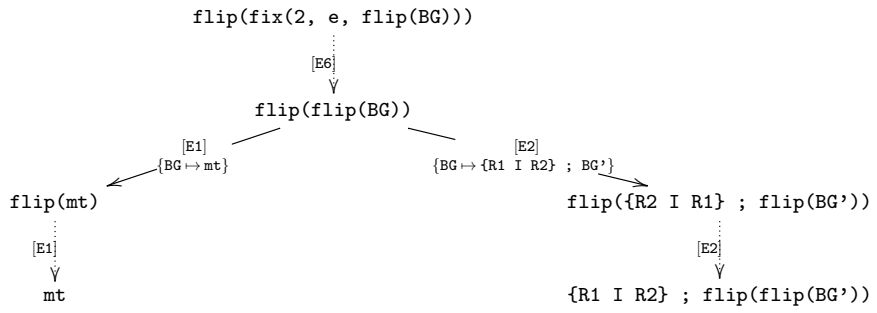

Figure 9: Folding variant narrowing tree for the goal `flip(fix(2, e, flip(BG)))`.

`fix` *symbol. All the leaves in the narrowing tree for t, shown in Figure 9, are B-closed w.r.t. the set of calls* {`flip(fix(2,e,flip(BG)))`,`flip(flip(BG'))`}. *This leads to the following specialized equations:*

```
eq flip(fix(2,e,flip(mt))) = mt .
eq flip(fix(2,e,flip({R1 I R2} ; BG))) = {R1 I R2} ; flip(flip(BG)) .
eq flip(flip(mt)) = mt .
eq flip(flip({R1 I R2} ; BG)) = {R1 I R2} ; flip(flip(BG)) .
```

The use of folding variant narrowing during partial evaluation provides good overall behavior regarding both the elimination of intermediate data structures and the propagation of information. Moreover, the following result establishes that the executability requirements imposed on the original theory are preserved by the transformation; e.g., no infinite o diverging computations are encoded in the residual program.

Theorem 4. *The PE of a decomposition* $(\Sigma, B, \vec{E})$ *is a decomposition.*

In the following section, we extend the classical post-processing transformation [14] to the order-sorted case modulo axioms to deliver a final partially evaluated program without any redundant or undesirable derivation that could not be proven in the original program.

3.6. Post-processing renaming modulo axioms

The basic PE algorithm of Section 3 incorporates only the basic scheme of a complete partial evaluator. The resulting partial evaluations can be further optimized by eliminating redundant function symbols and unnecessary repetition of variables. Essentially, we introduce a new function symbol for each specialized term and then replace each call in the specialized program by a call to the corresponding renamed function.

Definition 19 (Independent Renaming [14]). *An independent renaming ρ for a set of Σ-terms T is a mapping from terms to terms defined as follows: for $t \in T$ with $root(t) = f$ being a function symbol, $\rho(t) = f_t(\overline{x_n})$, where $\overline{x_n}$ are the distinct variables in t in the order of their first occurrence and f_t is a new function symbol, which does not occur in Σ or T and is different from the root symbol of any other $\rho(t')$, with $t' \in T$ and $t' \neq t$. By abuse, we let $\rho(T)$ denote the set $T' = \{\rho(t) \mid t \in T\}$.*

The renaming post-processing can be formally defined as follows.

Definition 20 (Post-partial Evaluation). *Let T be a finite set of terms and $\mathscr{R}'$ the (rewrite theory computed as a) partial evaluation of the rewrite theory $\mathscr{R}$ w.r.t. T s.t. $\mathscr{R}'$ is T-closed modulo B. Let ρ be an independent renaming for T. We define the post-partial evaluation $\mathscr{R}''$ of $\mathscr{R}$ w.r.t. T (under ρ) as follows:*

$$\mathscr{R}'' = \bigcup_{t \in T} \{\rho(t)\theta \to ren_\rho(r) \mid t\theta \to r \in \mathscr{R}'\}$$

where the nondeterministic renaming function ren_ρ is defined as follows:

$$ren_\rho(t) = \begin{cases} t & \text{if } t \in \mathscr{X} \\ c(\overline{ren_\rho(t_n)}) & \text{if } t = c(\overline{t_n}),\ c \in \mathscr{C} \\ \rho(u)\theta' & \text{if } \exists\theta, \exists u \in T \text{ such that } t =_B u\theta \text{ and} \\ & \quad \theta' = \{x \mapsto ren_\rho(x\theta) \mid x \in Dom(\theta)\} \\ t & \text{otherwise} \end{cases}$$

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

Theorem 5. *The post-partial evaluation of a decomposition $(\Sigma, B, \vec{E})$ is a decomposition.*

Finally, we state and prove the strong correctness of our partial evaluation technique. The proof proceeds essentially as follows. First, we prove the soundness (resp. completeness) of the transformation, i.e., we prove that, for each answer computed by folding variant narrowing in the original (resp. specialized) program, there exists a more general answer (modulo B) in the specialized (resp. original)

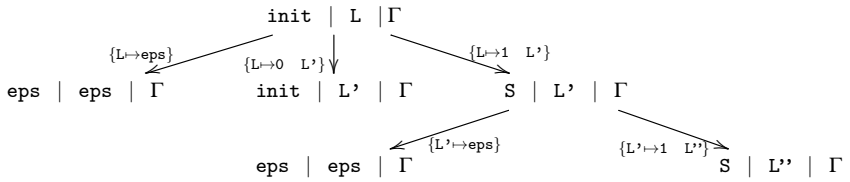

Figure 10: Folding variant narrowing tree for the goal `init | L | Γ`.

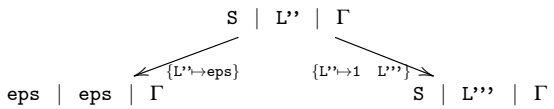

Figure 11: Folding variant narrowing tree for the goal `S | L'' | Γ`.

program. Then, by using the minimality of folding variant narrowing, we conclude the strong correctness of the method, i.e., the answers computed in the original and the partially evaluated programs coincide (modulo B).

Theorem 6 (Strong Correctness and Completeness of Post-partial Evaluation). *Let $\mathscr{R} = (\Sigma, B, \vec{E})$ be a decomposition of an equational theory $(\Sigma, E \uplus B)$, u be a Σ-term, and Q be a finite set of Σ-terms. Let ρ be an independent renaming of Q, $u' = ren_\rho(u)$ and $Q' = ren_\rho(Q)$. Let $\mathscr{R}' = (\Sigma, B, \vec{E}')$ be a partial evaluation of $\mathscr{R}$ w.r.t. Q (under the renaming ρ). If $\vec{E}'$ and u' are closed modulo B w.r.t. Q', then $(u \leadsto^*_{\sigma,\vec{E},B} v) \in VN^{\circlearrowleft}_{\mathscr{R}}(u)$ if and only if $(u' \leadsto^*_{\sigma',\vec{E}',B} v') \in VN^{\circlearrowleft}_{\mathscr{R}'}(u)$, where $v' =_B ren_\rho(v)$.*

Example 22 (Example 21 continued). *Consider the following independent renaming for the specialized calls:*

$$\{\texttt{flip(flip(BG))} \mapsto \texttt{dflip(BG)}\,,\ \texttt{flip(fix(2,e,flip(BG)))} \mapsto \texttt{flix(BG)}\}$$

The post-processing renaming derives the renamed program

```
eq flix(mt) = mt .
eq flix({R1 I R2} ; BG) = {R1 I R2} ; dflip(BG) .
eq dflip(mt) = mt .
eq dflip({R1 I R2} ; BG') = {R1 I R2} ; dflip(BG') .
```

Example 23. *Consider again the elementary parser defined in Example 1 and the initial configuration* `init | L | Γ`. *Following the PE algorithm, we construct*

the two folding variant narrowing trees that are shown in Figures 10 and 11. Now all leaves in the tree are closed w.r.t. Q and we get the following specialized parser:

```
eq init | eps   | Γ = eps  | eps | Γ [variant] .
eq init | 0 L   | Γ = init | L   | Γ [variant] .
eq init | 1     | Γ = eps  | eps | Γ [variant] .
eq init | 1 1 L | Γ = S    | L   | Γ [variant] .
eq S    | eps   | Γ = eps  | eps | Γ [variant] .
eq S    | 1 L   | Γ = S    | L   | Γ [variant] .
```

where the third and fourth equations are specialized versions of the following equation of Example 1

```
eq init | 1 L | Γ = S | L | Γ [variant] .
```

This is because the embedding test does not whistle until the expression `S | L' | Γ` *is reached.*

By applying the post-partial evaluation transformation with the independent renaming $\rho = \{$`init | L | Γ` $\mapsto$ `finit(L), S | L | Γ` $\mapsto$ `fS(L), eps | eps | Γ` $\mapsto$ `feps`$\}$*, we get the following specialized program (note that we obtain* `finit(1 eps) = feps`*, but it is simplified to* `finit(1) = feps` *modulo identity)*

```
eq finit(eps)   = feps .        eq finit(1) = feps .
eq finit(0 L)   = finit(L) .    eq fS(eps)  = feps .
eq finit(1 1 L) = fS(L) .       eq fS(1 L)  = fS(L) .
```

which gets rid of the grammar Γ *(and hence of costly* ACU-matching *operations) while still recognizing the string* `st` *by rewriting the simpler configuration* `finit(st)` *to the final configuration* `feps`*.*

4. Specializing the interpreter of an imperative programming language

As a final example, let us discuss several specializations of the interpreter of an imperative language whose implementation as a Maude equational theory is publicly available at the webpage of our tool, Victoria. The interpreter provides the standard semantics of a simple imperative language that transforms program configurations `P | M`, where `M` belongs to sort `Memory` and represents the program memory, and `P` is an imperative program that may contain assignment instructions, conditional statements, arithmetic expressions, and loops. For simplicity, the interpreter assumes that, in an initial configuration $\texttt{P}_0 \mid \texttt{M}_0$, the memory $\texttt{M}_0$ is initialized with default values for all the variables in the program $\texttt{P}_0$ and thus it contains pairs `[x,v]` for each program variable `x` in $\texttt{P}_0$.

Consider we want to specialize the interpreter w.r.t. the following input term configuration

```
x := 0 ; if (x = 0) then y := 0 fi ; skip | M          (Conf1)
```

where the Maude variable `M` stands for an unspecified program memory and is the only *logic or symbolic variable* in the input term, whereas program variables `x` and `y` are handled as constants by the interpreter.

Our tool Victoria returns the following extremely specialized version of the interpreter for the given input term, which can be seen as a compiled version written in Maude

```
eq x := 0 ; if (x = 0) then y := 0 fi ; skip | M [x,N1] [y,N2]
 = skip | M [x,0] [y,0] [variant] .
```

Given the independent renaming

$\rho = \{$"`x := 0 ; if (x = 0) then y := 0 fi ; skip | M`" $\mapsto$ `f1(M)`,
"`skip | M [x,0] [y,0]`" $\mapsto$ `f2(M)`$\}$

the final, renamed version of the program is

```
eq f1(M [x,N1] [y,N2]) = f2(M) [variant]  .
```

Let us now consider the case when the interpreter is specialized w.r.t. a more interesting configuration `P | M`, where there is a second logic variable `N` (in addition to `M`) that appears in `P` and belongs to sort `Nat` of natural numbers.

```
x := 2 ; i := N ; c := 0 ;
   while (i < x) do
      i := i + 1 ; c := c + i od ; skip                (Conf2)
| M
```

In this case, Victoria returns the following specialized interpreter

```
eq x := 2 ; i := 0 ; c := 0 ;
   while (i < x) do i := i + 1 ; c := c + i od ; skip
   | M [c,N1] [i,N2] [x,N3]
 = skip | M [c,3] [i,2] [x,2] [variant] .
eq x := 2 ; i := 1 ; c := 0 ;
   while (i < x) do i := i + 1 ; c := c + i od ; skip
   | M [c,N1] [i,N2] [x,N3]
 = skip | M [c,2] [i,2] [x,2] [variant] .
eq x := 2 ; i := 2 + N ; c := 0 ;
   while (i < x) do i := i + 1 ; c := c + i od ; skip
   | M [c,N1] [i,N2] [x,N3]
 = skip | M [c,0] [i,2 + N] [x,2] [variant] .
```

Given the independent renaming

$$\rho = \{\text{“x := 2 ; i := N; c := 0 ; while (i < x) do i := i + 1 ; c := c + i od ; skip | M”} \mapsto \texttt{f1(N,M)},$$
$$\text{“skip | M [c,0] [i,N] [x,2]”} \mapsto \texttt{f2(N,M)}\}$$

the final, renamed version of the program is

```
eq f1(0, M [c,N1] [i,N2] [x,N3]) = f2(0,M) [variant] .
eq f1(1, M [c,N1] [i,N2] [x,N3]) = f2(1,M) [variant] .
eq f1(2 + N, M [c,N1] [i,N2] [x,N3]) = f2(2 + N,M) [variant] .
```

Furthermore, if we make the memory of the previous input configuration more concrete (without any logic variable `M`)

```
x := 2 ; i := N ; c := 0;
 while (i < x) do i := i + 1 ; c := c + i od ; skip    (Conf3)
 | [c,0] [i,0] [x,0]
```

then our tool returns a simpler version of the same specialized program

```
eq f1(0) = f2(0) [variant] .
eq f1(1) = f2(1) [variant] .
eq f1(2 + N) = f2(2 + N) [variant] .
```

Note that this specialized program has no axiom, since the memory was defined as a multiset using an ACU symbol and it has been completely eliminated.

However, consider we specialize the interpreter w.r.t. the following symbolic configuration

```
x := N ; i := 0 ; c := 0;
  while (i < x) do i := i + 1 ; c := c + i od ; skip   (Conf4)
| [c,0] [i,0] [x,0]
```

where the logical variable `N` occurs in the assignment for the program variable `x` that controls the number of loop iterations. Then, our tool returns the following specialized version of the interpreter

```
eq x := 0 ; i := 0 ; c := 0;
   while (i < x) do i := i + 1 ; c := c + i od ; skip
   | [c,0] [i,0] [x,0]
```

```
 = skip | [c,0] [i,0] [x,0]  [variant] .
eq x := 1 + N ; i := 0 ; c := 0;
   while (i < x) do i := i + 1 ; c := c + i od; skip
   | [c,0] [i,0] [x,0]
 = if (i < x) then i := i + 1 ; c := c + i fi;
   while (i < x) do i := i + 1 ; c := c + i od ; skip
   | [c,1] [i,1] [x,1 + N] [variant] .
eq if (i < x) then i := i + 1 ; c := c + i fi ;
   while (i < x) do i := i + 1 ; c := c + i od ; skip
   | [c,N1] [i,N2 + 1 + N3] [x,N2] .
 = skip | [c,N1] [i,N2 + 1 + N3] [x,N2] [variant] .
eq if (i < x) then i := i + 1 ; c := c + i fi ;
   while (i < x) do i := i + 1 ; c := c + i od ; skip
   | [c,N1] [i,N2] [x,N2 + 1 + N3]
 = if (i < x) then i := i + 1 ; c := c + i fi ;
   while (i < x) do i := i + 1 ; c := c + i od ; skip
   | [c,N1 + N2] [i,N2 + 1] [x,N2 + 1 + N3] [variant] .
```

Given the independent renaming

$\rho = \{$“x := N; i := 0; c := 0;
while (i < x) do
i := i + 1; c := c + i od; skip
| [c,0] [i,0] [x,0]” $\mapsto$ f1(N),

“skip | M [c,N1] [i,N2] [x,N3]” $\mapsto$ f2(N1,N2,N3),

“if (i < x) then
i := i + 1; c := c + i fi;
while (i < x) do
i := i + 1; c := c + i od; skip
| [c,N1] [i,N2] [x,N3]” $\mapsto$ f3(N1,N2,N3)$\}$

the final, renamed version of the program is

```
eq f1(0) = f2(0,0,0) [variant] .
eq f1(1 + N) = f3(1,1,1 + N) [variant] .
eq f3(N1,N2 + 1 + N3,N2) = f2(N1,N2 + 1 + N3,N2) [variant] .
eq f3(N1,N2,N2 + 1 + N3) = f3(N1 + N2,N2 + 1,N2 + 1 + N3) [variant] .
```

Note that this specialization does not offer much improvement over the original interpreter, as expected because the while loop has been unrolled only once.

		Original	PE before renaming		PE after renaming	
Benchmark	**Data**	**Time (ms)**	**Time (ms)**	**Speedup**	**Time (ms)**	**Speedup**
Parser	100k	164	39	76,22	33	79,88
	1M	10.561	411	96,11	348	96,70
	5M	275.334	2.058	99,25	1.685	99,39
Double-flip	100k	188	143	23,94	76	59,57
	1M	1.636	1.427	12,78	759	53,61
	5M	8.425	7.503	10,94	4.100	51,34
Flip-fix	100k	203	177	12,81	143	29,56
	1M	1.955	1.778	9,05	1.427	27,01
	5M	10.185	9.219	9,48	7.458	26,77
KMP	100k	401	57	85,78	36	91,02
	1M	3.872	531	86,29	331	91,45
	5M	19.932	2.530	87,31	1.661	91,67
Interpreter	1k	5	3	40,00	2	60,00
	10k	53	22	58,49	12	77,36
	100k	520	248	52,30	112	78,46

Table 1: Experimental results

Interestingly, the specialization time is negligible in all these examples thanks to the (order-sorted) equational least general generalization of [7] and the homeomorphic embedding modulo equational axioms of [4]. This in contrast to our previous prototype tool in [5], which exceeded a generous timeout of several hours (similar to the specialization times for a comparable language interpreter in [49]). Furthermore, the size of the specialized program (after renaming) is less than 10% of the size of the original interpreter.

5. Experiments

We have implemented and experimentally evaluated the transformation framework presented in this article in the automatic partial evaluator Victoria[17] for Maude equational theories. Victoria has been implemented in Maude and consists of about ten thousand lines of code.

Table 1 contains the experiments that we have performed using an Intel Core2 Quad CPU Q9300(2.5GHz) with 6 Gigabytes of RAM running Maude v2.7.1 and considering the average of ten executions for each test. These experiments together with the source code of all examples are publicly available at Victoria's website. We have considered the four Maude programs previously discussed in the paper:

[17] Publicly available at `http://safe-tools.dsic.upv.es/victoria`.

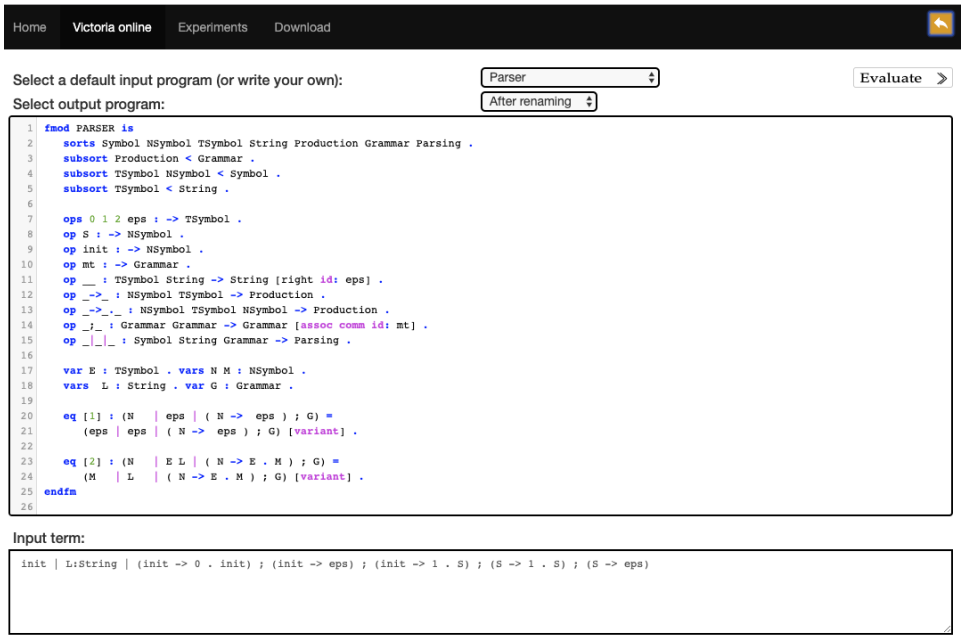

Figure 12: Specialization of the Parser example with Victoria

Parser (Example 1), Double-flip (Example 9), Flip-fix (Example 10), and an implementation of the interpreter for an imperative language (the example discussed in Section 4). We have also considered the classical KMP string pattern matcher [14]. For all five Maude programs, we consider input data of three different sizes: one hundred thousand elements, one million elements, and five million elements. Here elements refer to graph nodes for Double-flip and Flip-fix, list elements for Parser and KMP, and the value assigned to variable `x` in the configuration `Conf2` of Section 4 for Interpreter. We have benchmarked three versions of each program on these data: original program, partially evaluated program (before post-processing renaming), and final specialization (with post-processing renaming). We do not explicitly show the specialization times since they are negligible for all problems ($<$ 100 ms), which means it is up to 6 orders of magnitude faster than the preliminary prototype tool in [5].

The relative speedups that we achieved thanks to specialization are given in the Speedup column(s) and computed as the percentage $100 \times (\mathsf{OriginalTime} - \mathsf{PETime})/\mathsf{OriginalTime}$. For all of the examples, the partially evaluated programs achieve a significant improvement in execution time when compared to the original program, both with and without renaming, but even more noticeable after renaming. The average improvement for these benchmarks is 67.6%. Often, the price

Figure 13: Specialized Parser after renaming using Victoria

paid is the size of the residual program, which may grow linearly with the size of the specialized call. For the KMP test, the maximum improvement is 91.67%. That is, the achieved speedup is 12 (OriginalTime/PETime), which is comparable to the average speedup of 14 that is achieved by both the partial evaluator ECCE [43] and the PE tool of [3] (actually, the generated residual programs are identical to [3] on this benchmark). This shows that our new partial evaluation scheme is a conservative extension of previous approaches on comparable examples.

Moreover, matching modulo axioms such as associativity, commutativity, and identity are fairly expensive operations that are massively used in Maude, which can sometimes be drastically reduced after specialization. For instance, when we specialized the Parser of Example 1 using our tool Victoria, as illustrated in Figure 12, it moves from a program with ACU and U-right operators to a program without axioms, as illustrated in Figure 13; note that Victoria displays both the old and new signature with their axioms to help the reader understand the independent renaming. This transformation power cannot be achieved by traditional NPE nor competing on-line partial evaluation techniques, such as conjunctive partial deduction or positive supercompilation [15].

6. Conclusion and Future Work

Partial evaluation is a program optimization technique that is particularly effective at eliminating unnecessary overheads. We have laid the foundations for building the first fully automated partial evaluation system for Maude equational theories. Our specializer implements novel control criteria that ensure both local and global termination, which required classical partial evaluation notions to be correctly and efficiently generalized to the equational case; e.g., the order-sorted symbolic homeomorphic embedding modulo axioms and the order-sorted equational least general generalization that allow us to define equational closedness and abstraction. We have evaluated our practical implementation of the system, Victoria, and have shown that it specializes programs quickly and achieves significant increases in speed on realistic examples.

The development of a complete partial evaluator for the entire Maude language requires dealing with some features that are not considered in this work and experimenting with more refined heuristics that maximize the specialization power. Future implementation work will focus on: (i) extending the equational NPE framework to deal with more complex rewrite theories that may include (conditional) rules, equations, and axioms; and (ii) developing refined heuristics that can lead to further optimizations under mild assumptions (e.g., based on identifying subtheories that enjoy the finite variant property and other commonly occurring properties).

We believe that advancing current PE research ideas for order-sorted rewrite theories may open up new opportunities for optimization in rewriting logic program semantics development frameworks, e.g., [60]. Also, besides serving as a powerful tool to boost program performance, it will also be a significant driver of new symbolic reasoning features in Maude and further improvements in Maude's narrowing infrastructure.

Acknowledgments

We gratefully thank Demis Ballis and Julia Sapiña for many helpful discussions. We also thank the anonymous reviewers for their careful reading of our manuscript and their many insightful comments and suggestions.

References

References

[1] E. Albert, M. Alpuente, M. Falaschi, P. Julián, and G. Vidal. Improving Control in Functional Logic Program Specialization. In *Proc. of SAS '98*, pages 262–277. Springer LNCS 1503, 1998.

[2] E. Albert, M. Alpuente, M. Hanus, and G. Vidal. A Partial Evaluation Framework for Curry Programs. In H. Ganzinger, D. A. McAllester, and A. Voronkov, editors, *Proc. Logic Programming and Automated Reasoning, 6th International Conference, LPAR'99*, volume 1705 of *Lecture Notes in Computer Science*, pages 376–395. Springer, 1999.

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

[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 F. Calimeri, N. Leone, and M. Manna, editors, *Logics in Artificial Intelligence - 16th European Conference, JELIA 2019, Rende, Italy, May 7-11, 2019, Proceedings*, volume 11468 of *Lecture Notes in Computer Science*, pages 171–181. Springer, 2019.

[5] M. Alpuente, A. Cuenca-Ortega, S. Escobar, and J. Meseguer. Partial Evaluation of Order-Sorted Equational Programs Modulo Axioms. In *Proc. of 26th Int'l Symposium on Logic-Based Program Synthesis and Transformation, LOPSTR 2016*, volume 10184 of *LNCS*, pages 3–20. Springer, 2017.

[6] M. Alpuente, A. Cuenca-Ortega, S. Escobar, and J. Meseguer. Homeomorphic embedding modulo combinations of associativity and commutativity axioms. In F. Mesnard and P. J. Stuckey, editors, *Logic-Based Program Synthesis and Transformation - 28th International Symposium, LOPSTR 2018, Frankfurt/Main, Germany, September 4-6, 2018, Revised Selected Papers*, volume 11408 of *Lecture Notes in Computer Science*, pages 38–55. Springer, 2018.

[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*, 2019. To appear.

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

[9] M. Alpuente, S. Escobar, J. Meseguer, and P. Ojeda Order-Sorted Generalization. *Electr. Notes Theor. Comput. Sci.*, 246:27–38, 2009.

[10] M. Alpuente, S. Escobar, J. Meseguer, and P. Ojeda A Modular Equational Generalization Algorithm. In M. Hanus, editor, *Logic-Based Program Synthesis and Transformation - 18th International Symposium, LOPSTR 2008, Valencia, Spain, July 17-18, 2008, Revised Selected Papers*, volume 5438 of *Lecture Notes in Computer Science*, pages 24–39. Springer, 2009.

[11] M. Alpuente, S. Escobar, J. Sapiña, and A. Cuenca-Ortega. Inspecting maude variants with GLINTS. *TPLP*, 17(5-6):689–707, 2017.

[12] M. Alpuente, M. Falaschi, P. Julián, and G. Vidal. Specialization of Lazy Functional Logic Programs. In *Proc. of the ACM SIGPLAN Conf. on Partial Evaluation and Semantics-Based Program Manipulation, PEPM'97*, pages 151–162. ACM, New York, 1997.

[13] M. Alpuente, M. Falaschi, and G. Vidal. Narrowing-driven Partial Evaluation of Functional Logic Programs. In H. R. Nielson, editor, *Proc. of the 6th European Symp. on Programming, ESOP'96*, pages 45–61. Springer LNCS 1058, 1996.

[14] M. Alpuente, M. Falaschi, and G. Vidal. Partial Evaluation of Functional Logic Programs. *ACM TOPLAS*, 20(4):768–844, 1998.

[15] M. Alpuente, M. Falaschi, and G. Vidal. A Unifying View of Functional and Logic Program Specialization. *ACM Computing Surveys*, 30(3es):9es, 1998.

[16] F. Baader and W. Snyder. Unification theory. In J. A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 445–532. Elsevier and MIT Press, 2001.

[17] R. Bol. Loop Checking in Partial Deduction. *Journal of Logic Programming*, 16(1&2):25–46, 1993.

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

[19] M. Bruynooghe, D. De Schreye, and B. Martens. A General Criterion for Avoiding Infinite Unfolding during Partial Deduction of Logic Programs. In V. Saraswat and K. Ueda, editors, *Proc. 1991 Int'l Symp. on Logic Programming*, pages 117–131, 1991.

[20] H. Bürckert, A. Herold, and M. Schmidt-Schauß. On Equational Theories, Unification, and (Un)decidability. *Journal of Symbolic Computation*, 8(1–2):3–49, 1989.

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

[22] C. Cadar and K. Sen. Symbolic execution for software testing: Three decades later. *Commun. ACM*, 56(2):82–90, Feb. 2013.

[23] A. Cholewa, J. Meseguer, and S. Escobar. Variants of variants and the finite variant property. Technical report, CS Dept. University of Illinois at Urbana-Champaign, february 2014.

[24] N. H. Christensen and R. Glück. Offline partial evaluation can be as accurate as online partial evaluation. *ACM Trans. Program. Lang. Syst.*, 26(1):191–220, 2004.

[25] H. Comon-Lundh and S. Delaune. The finite variant property: How to get rid of some algebraic properties. In J. Giesl, editor, *RTA*, volume 3467 of *Lecture Notes in Computer Science*, pages 294–307. Springer, 2005.

[26] W. R. Cook and R. Lämmel. Tutorial on Online Partial Evaluation. In O. Danvy and C. Shan, editors, *Proc.IFIP Working Conference on Domain-Specific Languages, DSL 2011*, volume 66 of *EPTCS*, pages 168–180, 2011.

[27] O. Danvy, R. Glück, and P. Thiemann, editors. *Partial Evaluation, Int'l Seminar, Dagstuhl Castle, Germany*. Springer LNCS 1110, 1996.

[28] J. Darlington, Y. Guo, and H. Pull. Constraints unify functional and logic programming. Technical report, Department of Computing, Imperial College, London, 1991.

[29] N. Dershowitz and J.-P. Jouannaud. Rewrite Systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pages 243–320. Elsevier, Amsterdam, 1990.

[30] N. Dershowitz and J.-P. Jouannaud. Notations for Rewriting. *Bulletin of the European Association of Theoretical Computer Science*, 43:162–172, 1991.

[31] F. Durán, S. Eker, S. Escobar, N. Martí-Oliet, J. Meseguer, R. Rubio, and C. Talcott. Programming and symbolic computation in Maude. *J. Log. Algebr. Meth. Program.*, 2019. To appear.

[32] F. Durán, S. Eker, S. Escobar, N. Martí-Oliet, J. Meseguer, and C. L. Talcott. Associative unification and symbolic reasoning modulo associativity in Maude. In V. Rusu, editor, *Rewriting Logic and Its Applications - 12th International Workshop, WRLA 2018, Held as a Satellite Event of ETAPS, Thessaloniki, Greece, June 14-15, 2018, Proceedings*, volume 11152 of *Lecture Notes in Computer Science*, pages 98–114. Springer, 2018.

[33] F. Durán and J. Meseguer. A Maude coherence checker tool for conditional order-sorted rewrite theories. In P. C. Ölveczky, editor, *WRLA*, volume 6381 of *Lecture Notes in Computer Science*, pages 86–103. Springer, 2010.

[34] S. Eker. Associative-Commutative Rewriting on Large Terms. In R. Nieuwenhuis, editor, *Rewriting Techniques and Applications, 14th International Conference, RTA 2003, Proceedings*, volume 2706 of *Lecture Notes in Computer Science*, pages 14–29. Springer, 2003.

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

[36] S. Escobar, R. Sasse, and J. Meseguer. Folding variant narrowing and optimal variant termination. *J. Log. Algebr. Program.*, 81(7-8):898–928, 2012.

[37] M. Fay. First Order Unification in an Equational Theory. In *Proc of 4th Int'l Conf. on Automated Deduction, CADE'79*, pages 161–167, 1979.

[38] J. Goguen and J. Meseguer. Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105:217–273, 1992.

[39] M. Hanus. The Integration of Functions into Logic Programming: From Theory to Practice. *Journal of Logic Programming*, 19&20:583–628, 1994.

[40] M. Hanus. Functional Logic Programming: From Theory to Curry. In A. Voronkov and C. Weidenbach, editors, *Programming Logics - Essays in Memory of Harald Ganzinger*, volume 7797 of *Lecture Notes in Computer Science*, pages 123–168. Springer, 2013.

[41] M. Hanus and B. Peemöller. A partial evaluator for Curry. In *Proc. of 23rd International Workshop on Functional and (Constraint) Logic Programming (WFLP 2014)*, pages 55–71. Universität Halle-Wittenberg, 2014.

[42] N. Jones, C. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice-Hall, Englewood Cliffs, NJ, 1993.

[43] J. Jørgensen, M. Leuschel, and B. Martens. Conjunctive Partial Deduction in Practice. In *Proc. of LOPSTR'96*, pages 59–82. Springer LNCS 1207, 1996.

[44] J.-P. Jouannaud, C. Kirchner, and H. Kirchner. Incremental Construction of Unification Algorithms in Equational Theories. In *Proc. of 10th Colloquium on Automata, Languages and Programming (ICALP 1983)*, volume 154 of *LNCS*, pages 361–373. Springer, 1983.

[45] L. Lafave and J. P. Gallagher. Constraint-based partial evaluation of rewriting-based functional logic programs. In N. E. Fuchs, editor, *Proc. Logic Programming Synthesis and Transformation, 7th International Workshop, LOPSTR'97*, volume 1463 of *Lecture Notes in Computer Science*, pages 168–188. Springer, 1998.

[46] M. Leuschel. *Advanced Techniques for Logic Program Specialisation*. PhD thesis, 1997.

[47] M. Leuschel. Improving Homeomorphic Embedding for Online Termination. In P. Flener, editor, *Proc. of 8th International Workshop on Logic Programming Synthesis and Transformation, (LOPSTR 1998)*, volume 1559 of *LNCS*, pages 199–218. Springer, 1998.

[48] M. Leuschel and M. Bruynooghe. Logic program specialisation through partial deduction: Control issues. *TPLP*, 2(4-5):461–515, 2002.

[49] M. Leuschel, S. Craig, and D. Elphick. Supervising Offline Partial Evaluation of Logic Programs Using Online Techniques. In G. Puebla, editor, *Logic-Based Program Synthesis and Transformation, 16th International Symposium, LOPSTR 2006, Venice, Italy, July 12-14, 2006, Revised Selected Papers*, volume 4407 of *Lecture Notes in Computer Science*, pages 43–59. Springer, 2007.

[50] J. Lloyd and J. Shepherdson. Partial Evaluation in Logic Programming. *Journal of Logic Programming*, 11:217–242, 1991.

[51] B. Martens and J. Gallagher. Ensuring Global Termination of Partial Deduction while Allowing Flexible Polyvariance. In L. Sterling, editor, *Proc. of ICLP'95*, pages 597–611. MIT Press, 1995.

[52] P. Melliès. On a duality between Kruskal and Dershowitz theorem. In K. G. Larsen, S. Skyum, and G. Winskel, editors, *ICALP*, volume 1443 of *Lecture Notes in Computer Science*, pages 518–529. Springer, 1998.

[53] J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.

[54] J. Meseguer. Membership Algebra As a Logical Framework for Equational Specification. In F. Parisi-Presicce, editor, *Proc. of 12th International Workshop on Recent Trends in Algebraic Development Techniques, WADT'97*, volume 1376 of *LNCS*, pages 18–61. Springer, 1997.

[55] J. Meseguer. Order-Sorted Rewriting and Congruence Closure. In B. Jacobs and C. Löding, editors, *Foundations of Software Science and Computation Structures - 19th International Conference, FOSSACS 2016*, volume 9634 of *Lecture Notes in Computer Science*, pages 493–509. Springer, 2016.

[56] J. Meseguer. Strict Coherence of Conditional Rewriting Modulo Axioms. *Theor. Comput. Sci.*, 672:1–35, 2017.

[57] J. Meseguer. Variant-based satisfiability in initial algebras. *Sci. Comput. Program.*, 154:3–41, 2018.

[58] 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.

[59] G. Plotkin. A note on inductive generalization. In *Machine Intelligence*, volume 5, pages 153–163. Edinburgh University Press, 1970.

[60] V. Rusu, D. Lucanu, T. Serbanuta, A. Arusoaie, A. Stefanescu, and G. Rosu. Language definitions as rewrite theories. *J. Log. Algebr. Meth. Program.*, 85(1):98–120, 2016.

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

[62] P. Thati and J. Meseguer. Complete symbolic reachability analysis using back-and-forth narrowing. *Theor. Comput. Sci.*, 366(1-2):163–179, 2006.

[63] P. Viry. Equational rules for rewriting logic. *Theoretical Computer Science*, 285(2):487–517, 2002.

[64] P. Wadler. Deforestation: Transforming programs to eliminate trees. *Theoretical Computer Science*, 73:231–248, 1990.

[65] D. Weise, R. Conybeare, E. Ru, and S. Seligman. Automatic Online Partial Evaluation. In J. Hughes, editor, *FPCA*, volume 523 of *Lecture Notes in Computer Science*, pages 165–191. Springer, 1991.

[66] T. Würthinger, C. Wimmer, C. Humer, A. Wöß, L. Stadler, C. Seaton, G. Duboscq, D. Simon, and M. Grimmer. Practical Partial Evaluation for High-performance Dynamic Language Runtimes. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2017, pages 662–676, New York, NY, USA, 2017. ACM.

Appendix 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. We prove the following properties of our scheme:

1. (Section Appendix A.1) We prove that the closedness condition is reached by the generic Algorithm 1 independently from the narrowing strategy, unfolding rule, and abstraction operator.
2. (Section Appendix A.2) We prove the correctness and termination of the equational least general abstraction function.
3. (Section Appendix A.3) We prove that the use of the order-sorted symbolic homeomorphic embedding modulo B ensures termination of each iteration of Algorithm 1.
4. (Section Appendix A.4) We prove that the use of the equational least general abstraction function ensures termination of Algorithm 1.
5. (Section Appendix A.5) We prove that a partially evaluated program satisfies the same executability conditions imposed on the original program.
6. (Section Appendix A.6) We prove that a partially evaluated program is strongly correct and complete w.r.t. the folding variant narrowing computations of the original program.

Appendix A.1. B-closedness after EqNPE

We prove that the function EQNPE of Algorithm 1 reaches the closedness condition independently from the narrowing strategy, unfolding rule, and abstraction operator. In order to demonstrate this result, we first prove the transitivity of the equational closedness relation (Definifion 9).

We use the following auxiliary notion. We define the maximal number of nested symbols in a term t, $depth(t)$, as follows: $depth(X) = 1$, for $X \in \mathscr{X}$; $depth(t) = 1 + max(\{depth(t_1), ..., depth(t_n)\})$, for $t = f(t_1, ..., t_n), f \in \Sigma$, $n \geq 0$. The function *max* computes the maximum of a set on numbers.

Lemma 21. *If term t is B-closed w.r.t. a set T_1 of terms, and T_1 is B-closed w.r.t. a set T_2 of terms, then t is B-closed w.r.t. T_2.*

Proof. We proceed by induction on the depth of t.

Since the base case $depth(t) = 1$ is trivial, we proceed with the inductive case. Let $depth(t) = k + 1$, $k \geq 0$, and assume the property holds for all t' such that $depth(t') \leq k$. Since t is B-closed w.r.t. T_1 (and $depth(t) > 1$), following the definition of closedness we need to distinguish two cases:

1. If t is headed by a constructor symbol, then $t \equiv c(t_1,\ldots,t_n)$, $c \in \mathscr{C}$, and the set $\{t_1,\ldots,t_n\}$ is B-closed w.r.t. T_1. Since $depth(t_i) \leq k$, $i = 1,\ldots,n$, then by the induction hypothesis, $closed_B(T_2,t_i)$ holds for all $i = 1,\ldots,n$. Hence, by definition of closedness, t is also B-closed w.r.t. T_2.
2. If there exist $s_1 \in T_1$ and substitution θ_1 such that $root(t) = root(s_1) \in \mathscr{D}$ and $s_1\theta_1 =_B t$, then $closed_B(T_1,s')$ holds for all $x \mapsto s' \in \theta_1$. Since $s_1 \in T_1$, then by hypothesis we have that s_1 is B-closed w.r.t. T_2. Hence, there exists $s_2 \in T_2$ such that $s_2\theta_2 =_B s_1$ (since s_1 is also headed by a defined function symbol), and $closed_B(T_2,s'')$ holds for all $x \mapsto s'' \in \theta_2$. Then, we have that there is a term $s_2 \in T_2$ such that $s_2\theta_2\theta_1 =_B s_1\theta_1 =_B t$. To prove that t is B-closed w.r.t. T_2 we only need to show that $closed_B(T_2,t')$ holds for all $x \mapsto t'$ in $\theta_1\theta_2$. Since $depth(s') \leq k$ for all $x \mapsto s' \in \theta_1$, then by the induction hypothesis, we have that $closed_B(T_2,s')$ also holds for all $x \mapsto s' \in \theta_1$. Finally, since $closed_B(T_2,s'')$ holds for all $x \mapsto s'' \in \theta_2$, it is immediate that $x\theta_2\theta_1$ is B-closed w.r.t. T_2 for all $x \in Dom(\theta_2\theta_1)$, which ends the proof.

Lemma 11. *Let $\mathscr{R} = (\Sigma, B, \vec{E})$ be a decomposition of an equational theory $(\Sigma,\mathscr{E})$, $\mathscr{S}$ a narrowing strategy, and Q a set of terms. If* EQNPE$(\mathscr{R},Q,\mathscr{S})$ *terminates computing the set Q' of terms, then: (1) Q is B-closed w.r.t. Q', and (2) also the rules in the resulting partially evaluated theory $\mathscr{R}'$ are B-closed w.r.t. Q'.*

Proof. Consider the following sequence $Q_0,\cdots,Q_n$ of term sets that are computed at the successive iterations of the EQNPE function of Algorithm 1, where $Q_0 = Q$ and for all i, $1 \leq i \leq n$, we have $Q_i =$ ABSTRACT$(Q_{i-1},$UNFOLD$(Q_{i-1},\mathscr{R},\mathscr{S}),B)$.

Claim (1). Let us first prove that each Q_{i-1} is B-closed w.r.t. Q_i, for $1 \leq i \leq n$. From fact (2) of Definition 10 we have that, for $i > 0$, if $t \in Q_{i-1}$, then t is B-closed with respect to Q_i, and Claim (1) follows by transitivity (Lemma 21).

Claim (2). Since $Q_n =$ ABSTRACT$(Q_{n-1},$UNFOLD$(Q_{n-1},\mathscr{R},\mathscr{S}),B)$, by using again fact (2) from Definition 10, for all $t \in$ UNFOLD$(Q_{n-1},\mathscr{R},\mathscr{S})$, t is B-closed with respect to Q_n. Now, since $Q_n = Q_{n-1}$, the unfolding set UNFOLD$(Q_n,\mathscr{R},\mathscr{S})$ is B-closed w.r.t. Q_n, hence so are the right-hand sides of the rules in $\mathscr{R}'$ and the proof is done.

Appendix A.2. Correctness and termination of the abstraction function

In the following, we demonstrate the correctness and the termination abstraction function of Definition 17.

Let us define the complexity of a set of terms as follows.

Definition 22 (Complexity of a set of terms). *Let T be a set of terms. The complexity $\mathscr{M}_T$ of T is the finite multiset of natural numbers corresponding to the depth of the terms of T: $\mathscr{M}_T = \{depth(t) \mid t \in T\}$.*

The set of finite multisets over $\mathbb{N}$ is denoted by $M(\mathbb{N})$. We consider the well founded total ordering $\prec_{mul}$ over multisets by extending the well founded ordering $<$ on $\mathbb{N}$ to the set $M(\mathbb{N})$ of finite multisets over $\mathbb{N}$.

Definition 23 (Multiset ordering). *Given $M, M' \in M(\mathbb{N})$, $M \prec_{mul} M'$ if and only if there exist $X \subseteq M, X' \subseteq M'$ such that $M = (M' \setminus X') \cup X$ and, for all $n \in X$, there exists $n' \in X'$, such that $n < n'$, where $<$ is the standard strict partial order over $\mathbb{N}$.*

The set $M(\mathbb{N})$ is well founded under the *multiset ordering* $\prec_{mul}$ since $\mathbb{N}$ is well founded under $<$ [30].

Now, the desired result can be proved.

Proposition 18. *The function $abs^{\check{\trianglelefteq}_B}$ of Definition 17 is an abstraction operator in the sense of Definition 10.*

Proof. Following the definition of an abstraction operator (Definition 10), we need to prove that

1. if $s \in abs^{\check{\trianglelefteq}_B}(Q,T)$ then there exists $t \in (Q \cup T)$ such that $t_{|p} =_B s\theta$ for some position p and substitution θ and
2. for all $t \in (Q \cup T)$, t is B-closed w.r.t. $abs^{\check{\trianglelefteq}_B}(Q,T)$.

Condition (1) 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. Now we prove Condition (2) by well-founded induction on $\mathscr{M}_{Q \cup T}$.

If $Q \cup T = \emptyset$ ($\mathscr{M}_{Q \cup T} = \emptyset$), then $abs^{\check{\trianglelefteq}_B}(\emptyset,\emptyset) = \emptyset$, and the proof is done.

Let us consider the inductive case $Q \cup T \neq \emptyset$ ($\mathscr{M}_{Q \cup T} \neq \emptyset$), and assume that T is not empty (otherwise the proof is trivial). Then, $T = T_0 \cup \{t\}$, with $T_0 = \{t_1, \ldots, t_{n-1}\}$. By the inductive hypothesis, the property holds for all $Q^\star$ and for all $T^\star$ such that $\mathscr{M}_{Q^\star \cup T^\star} <_{mul} \mathscr{M}_{Q \cup T}$.

Following the definition of $abs^{\check{\trianglelefteq}_B}$, we now consider three cases:

1. If t is a variable symbol, or t is a constant value $c \in \mathscr{C}$, then

$$\begin{aligned} Q' &= abs^{\check{\trianglelefteq}_B}(Q,T) \\ &= abs^{\check{\trianglelefteq}_B}(Q,T_0 \cup \{t\}) \\ &= abs^{\check{\trianglelefteq}_B}(abs^{\check{\trianglelefteq}_B}(\ldots abs^{\check{\trianglelefteq}_B}(Q,t_1),\ldots,t_{n-1}),t) \\ &= abs^{\check{\trianglelefteq}_B}(abs^{\check{\trianglelefteq}_B}(Q,T_0),t) \\ &= abs^{\check{\trianglelefteq}_B}(Q,T_0). \end{aligned}$$

   Since $\mathscr{M}_{Q \cup T_0} <_{mul} \mathscr{M}_{Q \cup T_0 \cup \{t\}} = \mathscr{M}_{Q \cup T}$ then, by the inductive hypothesis, $Q \cup T_0$ is B-closed with respect to the terms in Q', and by the definition of B-closedness, so is $Q \cup T_0 \cup \{t\} = Q \cup T$.

2. If $t \equiv c(s_1,\ldots,s_m)$, $c \in \mathscr{C}$, $m > 0$, then by definition of $abs^{\breve{\trianglelefteq}_B}$

$$\begin{aligned} Q' &= abs^{\breve{\trianglelefteq}_B}(Q,T) \\ &= abs^{\breve{\trianglelefteq}_B}(Q,T_0 \cup \{t\}) \\ &= abs^{\breve{\trianglelefteq}_B}(abs^{\breve{\trianglelefteq}_B}(Q,T_0), c(s_1,\ldots,s_m)) \\ &= abs^{\breve{\trianglelefteq}_B}(abs^{\breve{\trianglelefteq}_B}(Q,T_0), \{s_1,\ldots,s_m\}) \\ &= abs^{\breve{\trianglelefteq}_B}(\ldots abs^{\breve{\trianglelefteq}_B}(abs^{\breve{\trianglelefteq}_B}(Q,T_0), s_1), \ldots, s_m) \\ &= abs^{\breve{\trianglelefteq}_B}(Q, T_0 \cup \{s_1,\ldots,s_m\}). \end{aligned}$$

Since $\mathscr{M}_{Q\cup T_0\cup\{s_1,\ldots,s_m\}} <_{mul} \mathscr{M}_{Q\cup T_0\cup\{c(s_1,\ldots,s_m)\}} = \mathscr{M}_{Q\cup T}$ then, by the inductive hypothesis, $Q \cup T_0 \cup \{s_1,\ldots,s_m\}$ is B-closed with respect to the terms in q', and by the definition of B-closedness, so is $Q \cup T_0 \cup \{c(s_1,\ldots,s_m)\} = Q \cup T$.

3. If $t \equiv f(s_1,\ldots,s_m)$, $f \in \mathscr{F}$, $m \geq 0$, then by definition of $abs^{\breve{\trianglelefteq}_B}$

$$\begin{aligned} Q' &= abs^{\breve{\trianglelefteq}_B}(Q,T) \\ &= abs^{\breve{\trianglelefteq}_B}(Q,T_0 \cup \{t\}) \\ &= abs^{\breve{\trianglelefteq}_B}(abs^{\breve{\trianglelefteq}_B}(Q,T_0), t) \\ &= generalize_B(abs^{\breve{\trianglelefteq}_B}(Q,T_0), W, t). \end{aligned}$$

where $W = \{t' \in abs^{\breve{\trianglelefteq}_B}(Q,T_0) \mid root(t) = root(t') \text{ and } t' \breve{\trianglelefteq}_B t\}$. Assume that $Q'' = abs^{\breve{\trianglelefteq}_B}(Q,T_0) \neq \emptyset$ (since the case when $abs^{\breve{\trianglelefteq}_B}(Q,T_0)$ is $\emptyset$ is straightforward). Here we distinguish three cases, which correspond to the three case values of the function $generalize_B$ in definition of $abs^{\breve{\trianglelefteq}_B}$, given by $generalize_B(Q'',W,t)$

(a) $Q'' \cup \{t\}$ when $W = \emptyset$. Since $\mathscr{M}_{Q\cup T_0} <_{mul} \mathscr{M}_{Q\cup T}$ then, by the inductive hypothesis, $Q \cup T_0$ is B-closed w.r.t. Q'', and therefore it is B-closed w.r.t. $Q' = Q'' \cup \{t\}$. Then, the claim follows trivially from the fact that t is B-closed w.r.t. $Q'' \cup \{t\}$.

(b) Q'' when t is B-closed w.r.t. Q''. Since $\mathscr{M}_{Q\cup T_0} <_{mul} \mathscr{M}_{Q\cup T}$ then, by the inductive hypothesis, $Q \cup T_0$ is B-closed w.r.t. Q'', and therefore $Q \cup T$ is B-closed w.r.t. Q''.

(c) $abs^{\breve{\trianglelefteq}_B}(W',S)$ where $W' = Q \setminus BMS_B(W,t)$ and $S = \{l \mid q \in BMS_B(W,t), \langle w, \{\theta_1,\theta_2\}\rangle \in lgg_B(\{q,t\}), x \in Dom(\theta_1 \cup \theta_2),\ l \in \{w, x\theta_1, x\theta_2\}\}$. By definition of B-closedness and lgg_B, given $\langle w, \{\theta_1,\theta_2\}\rangle \in lgg_B(\{q,t\})$, q and t are B-closed w.r.t. $\{w, x\theta_1, x\theta_2\}$ for $x \in Dom(\theta_1 \cup \theta_2)$. Therefore $BMS_B(W,t) \cup \{t\}$ is B-closed w.r.t. S. Note that $BMS_B(W,t) \subseteq W \subseteq Q$ and $W' \subseteq Q$. Therefore, $\mathscr{M}_{W'\cup S} <_{mul} \mathscr{M}_{Q\cup T_0\cup\{t\}} = \mathscr{M}_{Q\cup T}$ and, by the inductive hypothesis, $W' \cup S$ is B-closed w.r.t. $abs^{\breve{\trianglelefteq}_B}(W',S)$.

By Lemma 21, $BMS_B(W,t) \cup \{t\}$ is B-closed w.r.t. S and $W' \cup S$ is B-closed w.r.t. $abs^{\breve{\trianglelefteq}_B}(W',S)$ implies $BMS_B(W,t) \cup \{t\}$ is B-closed w.r.t. $abs^{\breve{\trianglelefteq}_B}(W',S)$ and the conclusion follows.

Theorem 2. *The equational least general abstraction function $abs^{\breve{\trianglelefteq}_B}$ terminates.*

Proof. The termination of the abstraction function $abs^{\breve{\trianglelefteq}_B}$ can be proved by well-founded induction of $\mathscr{M}_{Q \cup T}$. The proof is similar to the proof of Proposition 18.

Appendix A.3. Local termination of partial evaluation

In order to prove Lemma 14, we first prove that the order-sorted symbolic homeomorphic embedding relation $\breve{\trianglelefteq}$ (without axioms) of Definition 12 is a wqo on the set $\mathscr{T}_\Sigma(\mathscr{X})$. This is an easy consequence of Kruskal's Tree Theorem.

Lemma 24. *The order-sorted symbolic homeomorphic embedding relation $\breve{\trianglelefteq}$ of Definition 12 is a well-quasi ordering on the set $\mathscr{T}_\Sigma(\mathscr{X})$.*

Proof. The proof is similar to [46]. We need the following concept from [29] that we adapt to fixed-arity symbols. Let $\overset{<}{\sim}$ be a relation on a set $\mathscr{S}$ of symbols. Then the embedding extension of $\overset{<}{\sim}$ is a relation $\overset{<}{\sim}_{emb}$ on terms, constructed (only) from the symbols in $\mathscr{S}$, which is inductively defined as follows:

1. $t \overset{<}{\sim}_{emb} f(t_1,...,t_n)$ if $t \overset{<}{\sim}_{emb} t_i$ for some i;
2. $f(s_1,\ldots,s_n) \overset{<}{\sim}_{emb} g(t_1,\ldots,t_n)$ if $f \overset{<}{\sim} g$ and $\forall i \in \{1,\ldots,n\} : s_i \overset{<}{\sim}_{emb} t_i$.

We define the relation $\overset{<}{\sim}$ on the set $\mathscr{S} = (\Sigma \cup \mathscr{X})$ of symbols as the least relation satisfying:

1. $x \overset{<}{\sim} y$ if $x \in \mathscr{X}$, $y \in \mathscr{X}$, and $\lceil x \rceil = \lceil y \rceil$;
2. $f \overset{<}{\sim} f$ if $f \in \Sigma$.

This relation is a wqo on $\mathscr{S}$ (because Σ is finite). Therefore, by Higman-Kruskal's theorem (see e.g., [29]), its embedding extension to terms, $\overset{<}{\sim}_{emb}$, (which is by definition identical to $\breve{\trianglelefteq}$) is a wqo on $\mathscr{T}_\Sigma(\mathscr{X})$. □

Now, we are ready to prove that $\breve{\trianglelefteq}_B$ is a well-quasi ordering on the set $\mathscr{T}_\Sigma(\mathscr{X})$.

Lemma 14. *Given a* class-finite *theory (Σ, B), the order-sorted symbolic homeomorphic embedding relation $\breve{\trianglelefteq}_B$ is a well-quasi ordering on the set $\mathscr{T}_\Sigma(\mathscr{X})$.*

Proof. A binary relation $\succ$ is noetherian (i.e., well-founded) on a set X if and only if its dual relation $\preceq$ (defined as $u \preceq v$ iff $u \not\succ v$) is well on X, i.e., in every sequence $(x_i)_{i\in\mathbb{N}}$ of elements of X, there exist $i < j$ such that $x_i \preceq x_j$ [52]. Since $\breve{\trianglelefteq}$ is a wqo (by Lemma 24), the result follows for class-finite theories from (i) the fact that $>_B$ (i.e., the dual of $\leq_B$) is well-founded [20], and (ii) $\breve{\trianglelefteq}$ is compatible with $(\overset{ren}{=}_B)$. □

Now we are able to prove that our equational PE scheme always produces finite equational variant narrowing specializations.

Theorem 1. *Let $\mathscr{R} = (\Sigma, B, \vec{E})$ be a decomposition of an equational theory $(\Sigma, E \uplus B)$ and Q be a finite set of terms. The computation of $Unfold^{\breve{\trianglelefteq}_B}(Q, \mathscr{R})$ terminates.*

Proof. The proof is immediate from Lemma 14.

Nontermination of the PE algorithm can be caused not only by the generation of an infinite narrowing tree but also by never reaching the closedness condition, implying the infinite generation of finite narrowing trees. In the following, we demonstrate global termination of the PE algorithm.

Appendix A.4. Global termination of partial evaluation

We define the notion of a non-embedding set, which allows us to characterize sets that can be ordered as a sequence of terms that fulfill a well-quasi order.

Definition 25 (Non-embedding Set). *A finite set Q of terms is called* non-embedding *if all its comparable elements (i.e., rooted by the same operation symbol) can be ordered into a sequence $t_1, \ldots, t_n$ that satisfies*

$$\forall i, j \in \{1, \ldots, n\}, i < j \Rightarrow t_i \not\breve{\trianglelefteq}_B t_j$$

Lemma 26. *Given a non-embedding set Q of terms and a set T of terms, the set $abs^{\breve{\trianglelefteq}_B}(Q, T)$ of terms is non-embedding.*

Proof. It is immediate from Definition 17.

Theorem 3. *Algorithm 1 terminates for the unfolding function $Unfold^{\breve{\trianglelefteq}_B}$ and the equational least general abstraction function $abs^{\breve{\trianglelefteq}_B}$.*

Proof. Let $\mathscr{R}$ be a rewrite theory and t_0 be a term. Consider the following sequence $Q_0, \cdots, Q_n$ of term sets that are computed at the successive iterations of the function EQNPE of Algorithm 1, where $Q_0 = \{t_0\}$ and for all i, $1 \leq i \leq n$, we have $Q_i = abs^{\breve{\trianglelefteq}_B}(Q_{i-1}, Unfold^{\breve{\trianglelefteq}_B}(Q_{i-1}, \mathscr{R}))$. The proof follows directly from the following facts:

- The number of sets of incomparable terms (i.e., rooted by different operation symbols) which can be formed using a finite number of defined function symbols is finite.
- By Lemma 14 and Lemma 26, each Q_i is a non-embedding set.
- By Theorem 1, the computation of each set $Unfold^{\check{\trianglelefteq}_B}(Q_{i-1}, \mathscr{R})$, $i \geq 1$, terminates.
- By Theorem 2, the function $abs^{\check{\trianglelefteq}_B}$ terminates and, hence, each iteration of Algorithm 1 terminates.

Appendix A.5. Preservation of executability conditions

We first give proper definitions for a complete set of resultants of a folding variant narrowing tree and the related notion of (indiscriminate) complete specialization of (a decomposition of) an equational theory based on folding variant narrowing. Given the folding variant narrowing derivation $(t \rightsquigarrow^{+}_{\sigma,\vec{E},B} t') \in VN^{\circlearrowleft}_{\mathscr{R}}(t)$, recall the *resultant* of this derivation is $(t\sigma \Rightarrow t')$.

Definition 27 (Complete Resultant Set). *Let $\mathscr{R} = (\Sigma, B, \vec{E})$ be a decomposition of an equational theory $(\Sigma, \mathscr{E})$ and t_0 be a Σ-term. We define $CRS(t_0)$ as any* finite *subset of the resultants for all of the derivations in $VN^{\circlearrowleft}_{\mathscr{R}}(t_0)$ such that it is* complete*, i.e., for each $(t_0 \rightsquigarrow_{\sigma_1,\vec{E},B} t_1 \cdots t_{n-1} \rightsquigarrow_{\sigma_{n-1},\vec{E},B} t_n) \in VN^{\circlearrowleft}_{\mathscr{R}}(t_0)$, $n > 0$, there exists $0 < i \leq n$ such that $(t_0\sigma_{i-1} \Rightarrow t_i) \in CRS(t_0)$.*

Given a finite set Q of Σ-terms, by abuse we denote $CRS(Q) = \{CRS(t) \mid t \in Q\}$.

Definition 28 (B-closed Variant Narrowing Specialization (CVNS)). *Let $\mathscr{R} = (\Sigma, B, \vec{E})$ be a decomposition of an equational theory $(\Sigma, E \uplus B)$ and Q be a finite set of Σ-terms that is B-closed w.r.t. itself. We define a B-*closed variant narrowing specialization (CVNS) *of Q in $\mathscr{R}$ as the rewrite theory $\mathscr{R}' = (\Sigma, B, \vec{E}')$ with $\vec{E}'$ being a strict B-coherence completion of a complete set of resultants $CRS(Q)$ and, for every resultant $l \Rightarrow r \in CRS(Q)$, r is B-closed w.r.t. Q.*

Since each equational NPE produced by Algorithm 1 is a CVNS[18], it suffices to prove that every CVNS is a decomposition.

Lemma 29 (Any CVNS is a decomposition). *Let $\mathscr{R} = (\Sigma, B, \vec{E})$ be a decomposition of an equational theory $(\Sigma, E \uplus B)$ and Q be a finite set of Σ-terms that is*

[18]Note that we do not need to explicitly compute the strict B-coherence completion of the partially evaluated program since the specialized program will be automatically completed by Maude.

B-closed w.r.t. itself. Every CVNS $\mathscr{R}'$ of $\mathscr{R}$ w.r.t. Q is a decomposition of the equational theory $(\Sigma, E \uplus B)$.

Proof. Assume $\mathscr{R}' = (\Sigma, B, \vec{E}')$. Following Definition 1, we must prove that the set $\vec{E}'$ of rewrite rules is *convergent*, i.e., sort-decreasing, terminating, confluent, and strictly coherent modulo B. Sort-decreasingness and termination are straightforward, since $t \to^*_{\vec{E}',B} t'$ implies $t \to^*_{\vec{E},B} t'$. From Definition 28, strict B-coherence holds trivially for a CVNS. Confluence modulo B is a direct consequence of the fact that only minimal, *most-general* folding variant narrowing sequences are generated among all enabled narrowing sequences [36, Theorem 5], i.e., no overlaps are introduced at the root position in the left-hand sides of the computed resultants, for all $l \to r, l' \to r' \in CRS(Q)$, $\nexists \sigma : l\sigma =_B l'\sigma$. On the other hand, if there exist σ and p, with $p \neq \Lambda$, such that $(l_{|_p})\sigma =_B l'\sigma$ for resultants $l \to r, l' \to r' \in CRS(Q)$, then $(l_{|_p})\sigma \to^*_{\vec{E},B} (l[r']_p)\sigma$ and $l\sigma \to^*_{\vec{E},B} r\sigma$. By confluence of $\mathscr{R}$ modulo B, $\exists w, w' : (l[r']_p)\sigma \to^*_{\vec{E},B} w, r\sigma \to^*_{\vec{E},B} w', w =_B w'$. In the CVNS $\mathscr{R}'$, we have that $l\sigma \to_{\vec{E}',B} r\sigma$ and $(l_{|_p})\sigma \to_{\vec{E}',B} (l[r']_p)\sigma$. Since $\mathscr{R}'$ is B-closed w.r.t. Q, $\exists u, u' : (l[r']_p)\sigma \to^*_{\vec{E}',B} u, r\sigma \to^*_{\vec{E}',B} u', u =_B u'$.

Lemma 29 immediately implies the following results.

Theorem 4. *The PE of a decomposition $(\Sigma, B, \vec{E})$ is a decomposition.*

Theorem 5. *The post-partial evaluation of a decomposition $(\Sigma, B, \vec{E})$ is a decomposition.*

Appendix A.6. Strong correctness and completeness of Partial Evaluation

Let us prove strong correctness (resp. completeness) of the transformation, i.e., we prove that for each narrowing sequence computed by folding variant narrowing in the original (resp. specialized) program there exists a corresponding folding variant narrowing sequence in the specialized (resp. original) program up to renaming (modulo B). Let us first motivate some relevant issues about strong correctness and completeness.

Note that strong correctness is generally difficult in narrowing-driven partial evaluation (see [14] for details), since there might be Σ-terms that are reducible in $\vec{E}'$ by narrowing while not being reducible by narrowing in $\vec{E}$. For example, consider the following decomposition $(\Sigma, \emptyset, \vec{E})$ with

$$\vec{E} = \{ \begin{array}{lcl} f(x) & \to & 0 \\ g(x) & \to & x \end{array} \},$$

and let $Q = \{f(g(x)), f(x), g(0)\}$ be the set of input terms to be specialized. A partial evaluation of Q in R using ordinary narrowing (and stopping the unfolding of narrowing trees at depth one) may produce the following transformed program

$$\vec{E}' = \{ \begin{array}{rcl} f(g(x)) & \to & f(x) \\ f(g(x)) & \to & 0 \\ f(x) & \to & 0 \\ g(0) & \to & 0 \ \}. \end{array}$$

But now the term $f(g(Z))$ has more narrowing sequences in $\vec{E}'$ than in $\vec{E}$, i.e., $f(g(Z)) \leadsto_{id,\vec{E}'} 0$ and $f(g(Z)) \leadsto_{\{Z\mapsto 0\},\vec{E}'} f(0) \leadsto_{id,\vec{E}'} 0$ whereas only $f(g(Z)) \leadsto^*_{id,\vec{E}} 0$. Therefore, note that applying an independent renaming $f(g(x)) \to f'(x)$ is critical not only to remove unnecessary symbols but also to ensure that there is no possible confusion between terms of the original and the specialized programs (and their evaluations). However, thanks to using folding variant narrowing, the specialized version of $\vec{E}$ using our methodology is

$$\vec{E}'' = \{ \begin{array}{rcl} f(g(x)) & \to & 0 \\ f(x) & \to & 0 \\ g(0) & \to & 0 \ \} \end{array}$$

and folding variant narrowing returns only a simplification sequence $f(g(Z)) \leadsto^*_{id,\vec{E}''} 0$ (using either the first or the second equation in $\vec{E}''$) because the folding variant narrowing strategy always generates minimal, most general narrowing sequences (see Definitions 7 and 8).

Note that the specialized program must be closed in order to ensure that all terms in the specialized program are correctly evaluated. For example, consider the program $\vec{E}$ above and the set $Q = \{f(g(x))\}$ of input terms to be specialized. A partial evaluation of Q in $\vec{E}$ could be $\vec{E}''' = \{f(g(x)) \to f(x), f(g(x)) \to 0\}$ but this program is not closed w.r.t. Q, since the expression $f(x)$ cannot be rewritten to 0 in $\vec{E}'''$.

Lemma 30 (Strong Correctness and Completeness of PE). *Let $\mathscr{R} = (\Sigma, B, \vec{E})$ be a decomposition of an equational theory $(\Sigma, E \uplus B)$, u be a Σ-term, and Q be a finite set of Σ-terms. Let $\mathscr{R}' = (\Sigma, B, \vec{E}')$ be an equational NPE of $\mathscr{R}$ w.r.t. Q. If $\vec{E}'$ and u are closed modulo B w.r.t. Q, then $(u \leadsto^*_{\sigma,\vec{E},B} v) \in VN^{\circlearrowleft}_{\mathscr{R}}(u)$ if and only if $(u \leadsto^*_{\sigma',\vec{E}',B} v') \in VN^{\circlearrowleft}_{\mathscr{R}'}(u)$, where $v' =_B v$.*

Proof. Similar to the proof of Theorem 3.18 in [14], by replacing closedness with B-closedness and narrowing in a convergent TRS $\vec{E}$ with folding variant narrowing

in a decomposition $(\Sigma, B, \vec{E})$. The completeness result follows straightforwardly from Lemma 11 and the strong completeness of folding variant narrowing w.r.t. $(\vec{E},B)$-normalized substitutions in the decomposition $(\Sigma, B, \vec{E})$, proved in [36, Theorem 4], The strong correctness is immediate from the fact that, unlike ordinary narrowing, folding variant narrowing only computes minimal, most general narrowing sequences [36, Theorem 5].

In order to prove that strong correctness and completeness are not lost after renaming modulo B, we need the following.

Definition 31 (Overlap modulo B). *A term s overlaps modulo B a term t if there is a nonvariable subterm $s_{|u}$ of s such that $s_{|u}$ and t unify modulo B. If $s = t$, we require that t be unifiable modulo B with a proper nonvariable subterm of s.*

Definition 32 (Independence modulo B). *A set of terms S is independent modulo B if there are no terms s and t in S such that s overlaps t modulo B.*

The following lemma formalizes a fundamental technical property of independent sets of terms modulo B which allows us to prove that the specialized program does not compute additional solutions for an input term that could not be computed in the original program.

Lemma 33. *Let S be an independent set of terms, t an S-closed term modulo B, and $t_{|u}$ a subterm of t such that $u \in Pos_\Sigma(t)$. If $t_{|u}$ unifies modulo B with some term $s \in S$, then $s \leq_B t_{|u}$.*

Proof. Similar to the proof of Lemma 3.34 in [14], by replacing $\leq$ by $\leq_B$.

Renaming preserves B-closedness so that all of the original narrowing sequences are kept after the post-processing transformation. Also, renaming ensures independence modulo B of the specialized procedures so that no spurious narrowing derivations are introduced by the transformation.

Proposition 34 (B-closedness and B-independence after renaming). *Let $\mathscr{R} = (\Sigma, B, \vec{E})$ be a decomposition of an equational theory $(\Sigma, E \uplus B)$, Q be a finite set of Σ-terms, and u be a B-closed term w.r.t. Q. Let ρ be an independent renaming of Q, $u' = ren_\rho(u)$, and $Q' = \bigcup_{t \in Q} \{\langle t, ren_\rho(t)\rangle\}$. Let $\mathscr{R}' = (\Sigma, B, \vec{E}')$ be an equational NPE of $\mathscr{R}$ w.r.t. Q (under the renaming ρ). Then,*

1. *$\mathscr{A} = \{t' \mid \langle t, t'\rangle \in Q'\}$ is independent modulo B,*
2. *$\mathscr{R}' \cup \{u'\}$ is B-closed w.r.t. $\mathscr{A}$.*

Proof. It follows directly from Lemma 33.

Finally, the key result of this section follows.

Theorem 6. ***(Strong Correctness and Completeness of Post-partial Evaluation).*** *Let $\mathscr{R} = (\Sigma, B, \vec{E})$ be a decomposition of an equational theory $(\Sigma, E \uplus B)$, u be a Σ-term, and Q be a finite set of Σ-terms. Let ρ be an independent renaming of Q, $u' = ren_\rho(u)$ and $Q' = ren_\rho(Q)$. Let $\mathscr{R}' = (\Sigma, B, \vec{E}')$ be a partial evaluation of $\mathscr{R}$ w.r.t. Q (under the renaming ρ). If $\vec{E}'$ and u' are closed modulo B w.r.t. Q', then $(u \leadsto^*_{\sigma,\vec{E},B} v) \in VN^{\circlearrowleft}_{\mathscr{R}}(u)$ if and only if $(u' \leadsto^*_{\sigma',\vec{E}',B} v') \in VN^{\circlearrowleft}_{\mathscr{R}'}(u)$, where $v' =_B ren_\rho(v)$.*

Proof. Immediate by Lemma 30 and Proposition 34.