Verifying Higher-Order Functional Programs with Pattern-Matching Algebraic Data Types

C.-H. Luke Ong
Oxford University Computing Laboratory
lo@comlab.ox.ac.uk

Steven J. Ramsay
Oxford University Computing Laboratory
ster@comlab.ox.ac.uk

Abstract

Type-based model checking algorithms for higher-order recursion schemes have recently emerged as a promising approach to the verification of functional programs. We introduce *pattern-matching recursion schemes* (PMRS) as an accurate model of computation for functional programs that manipulate algebraic data-types. PMRS are a natural extension of higher-order recursion schemes that incorporate pattern-matching in the defining rules.

This paper is concerned with the following (undecidable) verification problem: given a correctness property φ, a functional program $\mathcal{P}$ (*qua* PMRS) and a regular input set $\mathcal{I}$, does every term that is reachable from $\mathcal{I}$ under rewriting by $\mathcal{P}$ satisfy φ? To solve the PMRS verification problem, we present a sound *semi-algorithm* which is based on model-checking and counterexample guided abstraction refinement. Given a no-instance of the verification problem, the method is guaranteed to terminate.

From an order-n PMRS and an input set generated by a regular tree grammar, our method constructs an order-n *weak* PMRS which over-approximates *only* the first-order pattern-matching behaviour, whilst remaining completely faithful to the higher-order control flow. Using a variation of Kobayashi's type-based approach, we show that the (trivial automaton) model-checking problem for weak PMRS is decidable. When a violation of the property is detected in the abstraction which does not correspond to a violation in the model, the abstraction is automatically refined by 'unfolding' the pattern-matching rules in the program to give successively more and more accurate weak PMRS models.

Categories and Subject Descriptors D.2.4 [*Software Engineering*]: Software/Program Verification; F.3.1 [*Logics and Meanings of Programs*]: Specifying and Verifying and Reasoning about Programs

General Terms Languages, Verification

1. Introduction

In the past decade, huge strides have been made in the development of finite-state and pushdown model checking for software verification. Though highly effective when applied to first-order, imperative programs such as C, these techniques are much less useful for higher-order, functional programs. In contrast, the two standard approaches to the verification of higher-order programs are *type-based program analysis* on the one hand, and *theorem-proving* and *dependent types* on the other. The former is sound, but often imprecise; the latter typically requires human intervention.

Recently, a model-checking approach, based on *higher-order recursion schemes* (HORS), has emerged as a verification methodology that promises to combine accurate analysis and push-button automation. HORS are a form of simply-typed lambda-calculus with recursion and uninterpreted function symbols that is presented as a grammar and used as a generator of (possibly infinite) trees. Ong showed that the trees generated by HORS have a decidable modal mu-calculus theory [14] and Kobayashi introduced a novel approach to the verification of higher-order functional programs by reduction to their model-checking problems [6].

This method has been applied successfully to the Resource Usage Verification Problem [3] (and, through it, to such problems as reachability and control-flow analysis) for a simply typed functional language with finite data-types and dynamic resource creation and resource access primitives. The method relies on the existence of certain transformations which, given a functional program and a resource usage specification, reduce the corresponding verification problem to the question of whether the computation tree of the program, generated by a HORS, satisfies a resource-wise specification encoded by an automaton on infinite trees. Despite the high worst-case time complexity of the modal mu-calculus model-checking problem for recursion schemes, which is n-EXPTIME complete for order-n schemes, an implementation of this approach, TRecS, performs remarkably well on realistic inputs [7].

From a verification perspective, a serious weakness of the HORS approach is its inability to naturally model functional programs with infinite data structures, such as integers and algebraic data-types. This severely limits the potential impact of this programme as functions defined by cases on algebraic data types are ubiquitous in functional programming.

A model of functional programs. Our first contribution is the introduction of *pattern-matching recursion schemes*, which are HORS extended with a notion of pattern matching. A PMRS is a kind of restricted term-rewriting system. We believe that PMRS have a very natural syntax into which large classes of functional programs can readily be translated. A typical rule, which is required to be well typed, has the shape:

$$F\, x_1 \cdots x_m\, p(y_1, \cdots, y_k) \;\longrightarrow\; t$$

where the variables $x_1, \cdots, x_m$ are (possibly higher-order) formal parameters of the non-terminal (or *defined operator*) F. The expression $p(y_1, \cdots, y_k)$, which takes the place of the final parameter, is a *pattern* constructed from terminal (or *constructor*) symbols and variables $y_1, \cdots, y_k$.

POPL'11, January 26–28, 2011, Austin, Texas, USA.

Example 1. The following PMRS defines a function $Merge$: $\mathbf{ablist} \to \mathbf{ablist} \to \mathbf{ablist}$ that merges two lists of `a` and `b` by recursively destructing them.

$$\begin{array}{lll} Merge\ x\ \texttt{nil} & \longrightarrow & x \\ Merge\ x\ (\texttt{cons a}\ y) & \longrightarrow & \texttt{cons a}\ (Merge\ y\ x) \\ Merge\ x\ (\texttt{cons b}\ y) & \longrightarrow & \texttt{cons b}\ (Merge\ y\ x) \end{array}$$

The patterns in the second argument position are used both to decompose compound data structures (so as to select the required components), and to determine control flow. Selected components are communicated to the right-hand side of the chosen rule by means of binding to the variables in the pattern.

Remark 1. Our work is not the first to propose a pattern-matching extension to HORS. A recent paper by Kobayashi, Tabuchi and Unno [9] introduces an extension of HORS called *higher-order multi-parameter tree transducers* (HMTT). HMTT model functions that may employ pattern matching but, in return, must satisfy a rigid type constraint. An HMTT function takes tree arguments of input sort `i` (which are trees that can only be destructed) and returns a tree of sort `o` (which are trees that can only be constructed). Pattern matching is only allowed on trees of sort `i`. Consequently HMTT functions are not compositional in the natural way. We believe our PMRS model to be both simpler and more natural.

A verification problem. This paper is concerned with the following verification problem. Given a correctness property φ, a functional program $\mathcal{P}$ (*qua* deterministic PMRS) and a regular set $\mathcal{I}$ of input (constructor) terms, does every term that is reachable from $\mathcal{I}$ under rewriting by $\mathcal{P}$ satisfy φ? It is straightforward to see that the problem is undecidable.

Example 2. Consider the PMRS $\mathcal{P}$ which, when started from $Main$ takes as input a list of natural numbers and returns the same list with all occurrences of the number zero removed. The defining rules of $\mathcal{P}$ are given by:

$$Main\ m \longrightarrow Filter\ Nz\ m$$

$$\begin{array}{l} If\ a\ b\ \texttt{true} \longrightarrow a \\ If\ a\ b\ \texttt{false} \longrightarrow b \end{array}$$

$$\begin{array}{l} Nz\ \texttt{z} \longrightarrow \texttt{false} \\ Nz\ (\texttt{s}\ n) \longrightarrow \texttt{true} \end{array}$$

$$\begin{array}{l} Filter\ p\ \texttt{nil} \longrightarrow \texttt{nil} \\ Filter\ p\ (\texttt{cons}\ x\ xs) \longrightarrow \\ \quad If\ (\texttt{cons}\ x\ (Filter\ p\ xs))\ (Filter\ p\ xs)\ (p\ x) \end{array}$$

The input set $\mathcal{I}$ is given by a regular tree grammar $\mathcal{G}$ (equivalently order-0 recursion scheme). The defining rules of $\mathcal{G}$ are:

$$S \longrightarrow ListN$$

$$\begin{array}{l} N \longrightarrow \texttt{z} \\ N \longrightarrow \texttt{s}\ N \end{array}$$

$$\begin{array}{l} ListN \longrightarrow \texttt{nil} \\ ListN \longrightarrow \texttt{cons}\ N\ ListN \end{array}$$

As usual, the start symbol of $\mathcal{G}$ is taken to be S. The correctness property φ is: "any outcome of the program is a list containing no zeros". This is easily expressible as a trivial automaton $\mathcal{A}$, whose definition is omitted.

An algorithmic solution. Our second contribution is a sound but incomplete semi-algorithm for solving the problem, which is based on a counterexample-guided abstraction refinement loop [2, 11]. The input to the algorithm consists of a PMRS $\mathcal{P}$ representing the program, a regular tree grammar $\mathcal{G}$ (equivalently an order-0 recursion scheme) representing the set $\mathcal{I}$ of possible inputs to the program and a trivial tree automaton $\mathcal{A}$ (which is to say, an automaton on infinite trees with a trivial acceptance condition) representing a specification φ of good behaviour. The algorithm proceeds according to the diagram in Figure 1.

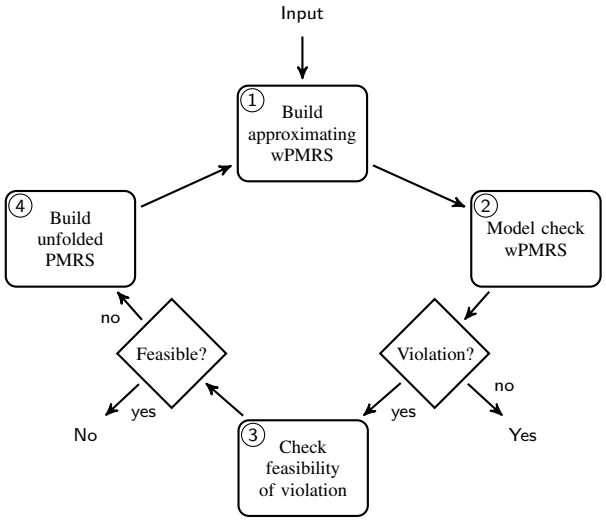


Figure 1. Counterexample-guided abstraction-refinement loop.

In step (1) we compute a sound abstraction of the behaviour of $\mathcal{P}$ when started from terms in $\mathcal{I}$. From an order-n PMRS $\mathcal{P}$ and an order-0 recursion scheme $\mathcal{G}$, we build an order-n *weak pattern-matching recursion scheme* (wPMRS) which over-approximates the set of terms that are reachable from $\mathcal{I}$ under rewriting by $\mathcal{P}$. A wPMRS is similar to a PMRS, except that its pattern matching mechanism is only able to determine control flow; it is unable to decompose data structure.

Our method is a kind of flow analysis. The first – and key – stage of the algorithm is a *binding analysis* which is inspired by Jones and Andersen [5]. It performs a fixpoint construction of a *finite* set Ξ of variable-term bindings such that, for every variable x (formal parameter of rewrite rule), every term that is ever bound to x during the computation is derivable from Ξ. In the second stage, we use the fixpoint set Ξ to build rules of the over-approximating wPMRS. These rules model the bindings of all non-pattern-matching (including all higher-order) variables *precisely*; they only approximate the binding behaviours of the pattern-matching variables. This is in contrast to Jones and Andersen's algorithm, which builds a regular tree grammar that over-approximates the binding set of *every* variable. For an order-n PMRS, our algorithm produces an order-n wPMRS $\widetilde{\mathcal{P}_{\mathcal{G}}}$ as an abstraction, which is a tighter approximation of the order-n PMRS being analysed than regular tree grammars (which are equivalent to order-0 wPMRS). To our knowledge, our algorithm gives the most accurate reachability / flow analysis of its kind.

The weakened pattern-matching mechanism of wPMRS makes it possible to decide a model checking problem for it, which is the content of step (2). Given a wPMRS $\mathcal{W}$, a closed term t and a Büchi automaton with a trivial acceptance condition $\mathcal{A}$, we decide if every (possibly infinite) tree generated by $\mathcal{W}$ on input t is accepted by $\mathcal{A}$. The proof uses a variation of Kobayashi's type-based approach.

If the model-checker fails to find any violation of the property then, since the approximating wPMRS $\widetilde{\mathcal{P}_{\mathcal{G}}}$ defines a superset of the terms reachable under $\mathcal{P}$ from $\mathcal{I}$, the loop in Figure 1 will terminate because $\mathcal{P}$ satisfies $\mathcal{A}$ on $\mathcal{I}$. However, if the model-checker reports a counterexample, then it may be that $\mathcal{P}$ also violates the property

(for some term in $\mathcal{I}$), but it may also be that the counterexample is an artifact of an inaccuracy in the abstraction. To determine which of these possibilities is the case, in step (3) we analyse the non-determinism introduced in the abstraction to see whether, in this particular counterexample, it behaves well or behaves badly.

In step (4) the abstraction process is refined. Due to the fact that the abstractions only ever approximate the (first-order) pattern matching variables, whilst remaining faithful to all the others, there is a simple notion of automatic abstraction-refinement, whereby patterns are "unfolded" to a certain depth in the PMRS $\mathcal{P}$, forming a new PMRS $\mathcal{P}'$. In the abstraction $\widetilde{\mathcal{P}'_{\mathcal{G}}}$ of $\mathcal{P}'$, the rules that define the approximation will be more accurate and, in particular, the spurious counterexample will no longer be present. Since any rule in a wPMRS abstraction $\widetilde{\mathcal{P}'_{\mathcal{G}}}$ is *perfectly* accurate whenever the pattern parameter contains no free variables, this method of unfolding gives rise to a semi-completeness property. Given any no-instance of the PMRS verification problem, the loop in Figure 1 will eventually terminate with the answer "No".

Returning to Example 2, whilst performing step (1) we obtain an over-approximation of the binding behaviour of the variables in the program Ξ. This fixpoint set contains, amongst others, the bindings: $x \mapsto N$ and $xs \mapsto ListN$. From this set, we construct an approximating wPMRS $\widetilde{\mathcal{P}_{\mathcal{G}}}$, whose rule-set contains the following:

$$\begin{array}{l} \mathit{Filter}\ p\ \mathtt{nil} \longrightarrow\ \ \mathit{Nil} \\ \mathit{Filter}\ p\ (\mathtt{cons}\ x\ xs) \longrightarrow \\ \quad \mathit{If}\ (\mathit{Cons}\ X\ (\mathit{Filter}\ p\ XS))\ (\mathit{Filter}\ p\ XS)\ (p\ X) \end{array}$$

$$\begin{array}{l} X \longrightarrow N \\ XS \longrightarrow ListN \end{array}$$

together with, amongst others, all the $\mathcal{P}$ and $\mathcal{G}$ rules in Example 2 except those for Filter. Unfortunately the wPMRS is too coarse to be useful: there are trees (representing lists) that are obtained by rewriting from '$\mathit{Main}\ S$' that are not accepted by the trivial automaton $\mathcal{A}$. However, these are spurious counterexamples. For an illustration, consider the error trace in the wPMRS:

$$\begin{array}{l} \mathit{Main}\ S \\ \rightarrow^* \mathit{Main}\ (\mathtt{cons}\ (\mathtt{s}\ \mathtt{z})\ \mathtt{nil}) \\ \rightarrow^* \mathit{Filter}\ \mathit{Nz}\ (\mathtt{cons}\ (\mathtt{s}\ \mathtt{z})\ \mathtt{nil}) \\ \rightarrow\ \ \mathit{If}\ (\mathit{Cons}\ X\ (\mathit{Filter}\ \mathit{Nz}\ \mathit{XS}))\ (\mathit{Filter}\ \mathit{Nz}\ \mathit{XS})\ (\mathit{Nz}\ X) \\ \rightarrow^* \mathit{If}\ (\mathtt{cons}\ \mathtt{z}\ (\mathit{Filter}\ \mathit{Nz}\ \mathtt{nil}))\ (\mathit{Filter}\ \mathit{Nz}\ \mathit{XS})\ (\mathit{Nz}\ (\mathtt{s}\ \mathtt{z})) \\ \rightarrow^* \mathtt{cons}\ \mathtt{z}\ (\mathit{Filter}\ \mathit{Nz}\ \mathtt{nil}) \\ \rightarrow^* \mathtt{cons}\ \mathtt{z}\ \mathtt{nil} \end{array}$$

The problem can be traced to the second clause of Filter in the wPMRS: when replacing the *variable* x by the *non-terminal* X, the connection between the two occurrences of x in the RHS is lost, as the reduction of one occurrence of X is independent of that of the other.

The refinement algorithm produces a new, unfolded PMRS $\mathcal{P}'$ that replaces the two defining rules of Filter by five new rules. The two rules that cover the case when the list is a singleton are shown below:

$$\begin{array}{l} \mathit{Filter}\ p\ (\mathtt{cons}\ \mathtt{z}\ \mathtt{nil}) \longrightarrow \\ \quad \mathit{If}\ (\mathtt{cons}\ \mathtt{z}\ (\mathit{Filter}\ p\ \mathtt{nil}))\ (\mathit{Filter}\ p\ \mathtt{nil})\ (p\ \mathtt{z}) \\ \mathit{Filter}\ p\ (\mathtt{cons}\ (\mathtt{s}\ v_2)\ \mathtt{nil}) \longrightarrow \\ \quad \mathit{If}\ (\mathtt{cons}\ (\mathtt{s}\ v_2)\ (\mathit{Filter}\ p\ \mathtt{nil}))\ (\mathit{Filter}\ p\ \mathtt{nil})\ (p\ (\mathtt{s}\ v_2)) \end{array}$$

Applying the approximation algorithm to PMRS $\mathcal{P}'$ (and input grammar $\mathcal{G}$), we obtain a wPMRS $\widetilde{\mathcal{P}'_{\mathcal{G}}}$ that does accurately capture the set of reachable terms.

$$\frac{}{\Gamma,\ x:\sigma\ \vdash\ x:\sigma}\ (\textsc{Var}) \qquad \frac{\xi:\sigma \in \Sigma \cup \mathcal{N}}{\Gamma\ \vdash\ \xi:\sigma}\ (\textsc{Const})$$

$$\frac{\Gamma\ \vdash\ t_0:\sigma\to\tau \qquad \Gamma\ \vdash\ t_1:\sigma}{\Gamma\ \vdash\ t_0\ t_1:\tau}\ (\textsc{App})$$

Figure 2. A simple type system for applicative terms.

Outline. The rest of the paper is organised as follows. Section 2 introduces PMRS, wPMRS and other technical preliminaries. In Section 3, the abstraction algorithm, which takes a program (PMRS) and an input set (order-0 recursion scheme) and returns a wPMRS, is presented; termination and soundness of the approximation are proved. Section 4 presents a type-inference algorithm for deciding if every tree generated by a given wPMRS is accepted by a trivial automaton. The abstraction refinement algorithm is the topic of Section 5. Finally Section 6 presents related work. Note: a long version of the paper is available [15], which contains the proofs and additional material.

2. Preliminaries

We introduce PMRS, a model for functional programs manipulating algebraic data types; wPMRS, a restriction of PMRS with good algorithmic properties and the PMRS Verification Problem, whose solution is the subject of the remainder of this work.

2.1 Types, terms and substitutions

Fix a finite set $(b,\ o\ \in)\ \mathbb{B}$ of *base types*. The *simple types* $(\sigma,\ \tau \in)\ \mathbb{S}$ are those expressions that can be constructed from the base types using the arrow:

$$\sigma,\ \tau\ \ ::=\ \ b\ \mid\ \sigma \to \tau.$$

We adopt the usual convention that arrows associate to the right and omit parenthesis accordingly. The *order* of a type τ, denoted $\mathbf{ord}(\tau)$, is a measure of the nestedness of the arrow constructor on the left; it is defined by $\mathbf{ord}(b) = 0$ and $\mathbf{ord}(\sigma \to \tau) = \mathsf{max}\{\mathbf{ord}(\sigma)+1,\ \mathbf{ord}(\tau)\}$.

Applicative terms. Fix a finite, simply-typed alphabet $(f, g, a \in)$ Σ of first-order *terminal symbols* (or *constructors*), a finite, simply-typed alphabet $(F, G, H\ \in)\ \mathcal{N}$ of (arbitrary-order) *non-terminal symbols* (or *defined operators*) and a denumerable set $(x,\ y,\ z\ \in)$ $\mathcal{V}$ of *variables*.

- The *constructor terms* $T(\Sigma)$ are those expressions that can be built from terminals using application.
- The *closed terms* $T(\Sigma,\ \mathcal{N})$ are those expressions that can be built from terminals and non-terminals using application.
- The *patterns* are those expressions $p,\ q$ of base type that can be built from variables of base type and terminals.
- The *applicative terms* $T(\Sigma,\ \mathcal{N},\ \mathcal{V})$ are those expressions that can be built from terminals, non-terminals and variables using application.

We denote the free variables of a term t by $\mathsf{FV}(t)$.

Standardly, applicative terms may be assigned simple types via a formal system of typing judgements, $\Gamma\ \ \vdash\ \ s\ :\ \tau$ (where Γ is a finite set of *type bindings*) defined by the rules in Figure 2. When an applicative term t can be assigned a simple type $\tau_1 \to \cdots \to \tau_m \to b$ we say that it has *arity* m and write $\mathsf{ar}(t) = m$. Henceforth, by *term* we shall mean well-typed, applicative term.

Labelled trees. Given a ranked alphabet Ω, an Ω-*labelled tree* t is a map from $\{\,1, \cdots, m\,\}^*$ to Ω, where m is the largest arity of

symbols in Ω, such that $\mathbf{dom}(t)$ is prefix-closed, and if $t(x) = f$ then $\{\, i \mid x\,i \in \mathbf{dom}(t) \,\} = \{\, 1, \cdots, \mathsf{ar}(f) \,\}$. Standardly we identify $T(\Sigma)$ with finite Σ-labelled trees, and write $T^{\infty}(\Sigma)$ for the collection of (possibly infinite) Σ-labelled trees.

Let $\Sigma^{\perp}$ be $\Sigma \cup \{\, \perp \,\}$ with $\mathsf{ar}(\perp) = 0$. Given a closed term t, we write $t^{\perp}$ for the finite, $\Sigma^{\perp}$-labelled tree defined by recursion as follows: for $m \geq 0$

$$(\xi\ s_1\ \cdots\ s_m)^{\perp} := \begin{cases} \perp & \text{if } \xi = F \in \mathcal{N} \\ f\ s_1^{\perp} \cdots s_m^{\perp} & \text{otherwise } \xi = f \in \Sigma \end{cases}$$

E.g. $(f(g(G\,a))b)^{\perp} = f(g\,\perp)b$. $\Sigma^{\perp}$-labelled trees can be endowed with a natural complete partial order $\sqsubseteq$ in which, for all trees t, $\perp \sqsubseteq t$ and $f\ s_1\ \cdots\ s_m \sqsubseteq f\ t_1\ \cdots\ t_m$ iff for all i, $s_i \sqsubseteq t_i$.

Substitutions. A *substitution* is just a partial function θ in $\mathcal{V} \to T(\Sigma, \mathcal{N}, \mathcal{V})$. By convention, we do not distinguish between a substitution and its homomorphic extension to the free algebra $T(\Sigma, \mathcal{N}, \mathcal{V})$ and we will write the application of both using prefix juxtaposition. A term t is said to *match* a term u precisely when there exists a substitution θ such that $t = \theta u$. We shall say that a substitution θ is *closed* whenever every term in its image is closed.

2.2 Pattern-matching recursion scheme (PMRS)

A *pattern-matching recursion scheme* (PMRS) is a quadruple $\mathcal{P} = \langle \Sigma, \mathcal{N}, \mathcal{R}, \mathit{Main} \rangle$ with Σ and $\mathcal{N}$ as above. $\mathcal{R}$ is a finite set of rewrite rules, each of which is one of the following shapes ($m \geq 0$):

$$\begin{array}{rl} (\textit{pure}) & F\ x_1\ \cdots\ x_m \longrightarrow t \\ (\textit{pattern-matching}) & F\ x_1\ \cdots\ x_m\ p \longrightarrow t \end{array}$$

where p is a pattern (which may be trivial). $\mathit{Main} : b \to o$ is a distinguished non-terminal symbol whose defining rules are always pattern-matching rules. In this paper we will assume that the variables appearing as formal parameters to defining rules in a PMRS will always be distinct.

A pure rule $F\ x_1\ \cdots\ x_m \longrightarrow t$ is well-typed when $F : \tau_1 \to \cdots \to \tau_m \to o \in \mathcal{N}$ and the judgement:

$$x_1 : \tau_1, \ldots, x_m : \tau_m \ \vdash\ t : o$$

is provable. A pattern-matching rule $F\ x_1\ \cdots\ x_m\ p \longrightarrow t$ is well-typed when $F : \tau_1 \to \cdots \to \tau_m \to b \to o \in \mathcal{N}$ and there exist base-types $b_1, \ldots, b_k$ such that the judgements:

$$y_1 : b_1, \ldots, y_k : b_k \ \vdash\ p : b \qquad \text{and}$$
$$x_1 : \tau_1, \ldots, x_m : \tau_m, y_1 : b_1, \ldots, y_k : b_k \ \vdash\ t : o$$

are provable. We say that a PMRS is *well-typed* just when each of its rules is well-typed. We will only consider well-typed PMRS in the following.

We define the *order* of a PMRS to be the maximum order of (the type of) any of the non-terminal symbols in $\mathcal{N}$. Since a pure rule can be simulated by a pattern-matching rule with a trivial pattern (e.g. a nullary terminal of a distinguished base type), we shall sometimes find it convenient to treat all PMRS rules as pattern-matching rules.

Reduction. We associate with each PMRS a notion of reduction as follows. A *redex* is a term of the form $F\ \theta x_1\ \cdots\ \theta x_m\ \theta p$ whenever θ is a closed substitution and $F\ x_1\ \cdots\ x_m\ p \longrightarrow t$ is a rule in $\mathcal{P}$. The *contractum* of the redex is $\theta\, t$. We define the one-step reduction relation, $\Rightarrow \subseteq T(\Sigma, \mathcal{N}) \times T(\Sigma, \mathcal{N})$, by $C[s] \Rightarrow C[t]$ whenever s is a redex, t is its contractum and C is a one-hole context.

We say that a PMRS is *deterministic* just if, given some redex $F\ s_1\ \cdots\ s_n$ there is exactly one rule $l \longrightarrow r \in \mathcal{R}$ such that $F\ s_1\ \cdots\ s_n = \theta l$ for some θ.

Given a PMRS $\mathcal{P} = \langle \Sigma, \mathcal{N}, \mathcal{R}, \mathit{Main} \rangle$, let $s \in T(\Sigma, \mathcal{N})$ be a closed term of base type. We write $\mathcal{L}(\mathcal{P}, s)$ to mean the language of $\Sigma^{\perp}$-labelled trees obtained by infinitary rewriting of the term s. More precisely, define $\mathcal{L}(\mathcal{P}, s)$ as the collection of $\Sigma^{\perp}$-labelled trees t such that there are $\langle t_i \rangle_{i \in \omega}$ with $s \Rightarrow t_1 \Rightarrow t_2 \Rightarrow t_3 \cdots$ a *fair* reduction sequence (in the sense that for each i, every outermost redex in t_i is eventually contracted) and $t = \bigsqcup \{\, t_i^{\perp} \mid i \in \omega \,\}$. In case $\mathcal{P}$ is a deterministic PMRS, $\mathcal{L}(\mathcal{P}, s)$ is a singleton set; we write the unique $\Sigma^{\perp}$-labelled tree as $[\![s]\!]_{\mathcal{P}}$.

Example 3. Let $\Sigma = \{\mathtt{zero} : \mathbf{nat}, \mathtt{succ} : \mathbf{nat} \to \mathbf{nat}, \mathtt{nil} : \mathbf{natlist}, \mathtt{cons} : \mathbf{nat} \to \mathbf{natlist} \to \mathbf{natlist}\}$ and $\mathcal{N} = \{\mathit{Rev} : \mathbf{natlist} \to \mathbf{natlist}, \mathit{RevA} : \mathbf{natlist} \to \mathbf{natlist} \to \mathbf{natlist}\}$. The following deterministic, order-1 PMRS contains rewrite rules that implement list reversal with an accumulating parameter:

$$\begin{array}{rcl} \mathit{Main}\ zs & \longrightarrow & \mathit{RevA}\ \mathtt{nil}\ zs \\ \\ \mathit{RevA}\ xs\ \mathtt{nil} & \longrightarrow & xs \\ \mathit{RevA}\ xs\ (\mathtt{cons}\ y\ ys) & \longrightarrow & \mathit{RevA}\ (\mathtt{cons}\ y\ xs)\ ys \end{array}$$

When started from the term $t = \mathtt{cons\ z\ nil}$, the only possible reduction sequence is:

$$\mathit{Main}\ t \Rightarrow \mathit{RevA}\ \mathtt{nil}\ t \Rightarrow \mathit{RevA}\ t\ \mathtt{nil} \Rightarrow t$$

and hence $[\![\mathit{Main}\ t]\!]_{\mathcal{P}} = t$, as expected.

2.3 Weak pattern matching recursion schemes (wPMRS)

A *weak pattern-matching recursion scheme* (wPMRS) is a quadruple $\mathcal{W} = \langle \Sigma, \mathcal{N}, \mathcal{R}, \mathit{Main} \rangle$ with $\Sigma, \mathcal{N}$ and Main as for PMRS. The (finite) set $\mathcal{R}$ consists of rewrite rules of the shape ($m \geq 0$):

$$\begin{array}{rl} (\textit{pure}) & F\ x_1\ \cdots\ x_m \longrightarrow t \\ (\textit{weak-matching}) & F\ x_1\ \cdots\ x_m\ p \longrightarrow t \end{array}$$

in which $\mathsf{FV}(p) \cap \mathsf{FV}(t) = \emptyset$. A pure rule is well typed according to the same criteria as for pure PMRS rules. A weak-matching rule $F\ x_1\ \cdots\ x_m\ p \longrightarrow t$ is well-typed just when $F : \tau_1 \to \cdots \to \tau_m \to b \to o \in \mathcal{N}$ and there exist base-types $b_1, \ldots, b_k$ such that the judgements:

$$y_1 : b_1, \ldots, y_k : b_k \ \vdash\ p : b$$
$$\text{and} \quad x_1 : \tau_1, \ldots, x_m : \tau_m \ \vdash\ t : o$$

are provable (note that none of the pattern-matching variables y_j occurs in t). Henceforth we will only consider wPMRS with well-typed rules.

wPMRS have exactly the same notion of reduction as PMRS: a *redex* is a term of the form $F\ \theta x_1\ \cdots\ \theta x_m\ \theta p$ whenever θ is a substitution and $F\ x_1\ \cdots\ x_m\ p \longrightarrow t$ is a rule in $\mathcal{P}$. The *contractum* of the redex is $\theta\, t = t[\theta\, x_1/x_1] \cdots [\theta\, x_m/x_m]$ (as the pattern-matching variables do not occur in t). The one-step reduction relation, $\rightarrow$, is defined as for PMRS.

We define the order, determinism and language of a wPMRS analogously with PMRS.

2.4 A verification problem

We are interested in solving the following verification problem. Given a program in the form of a PMRS $\mathcal{P}$, a regular set $\mathcal{I}$ of "input" terms, and a correctness property φ, does the output $[\![\mathit{Main}\ t]\!]$ of the program '$\mathit{Main}\ t$' satisfy φ, for every input $t \in \mathcal{I}$? To propose a solution, we require two further stipulations, both of which concern the representation of the entities involved.

Higher-order recursion schemes. A *higher-order recursion scheme* (HORS) is a quadruple $\mathcal{G} = \langle \Sigma, \mathcal{N}, \mathcal{R}, S \rangle$ with Σ and $\mathcal{N}$ as before and $\mathcal{R}$ is a finite set of well-typed, pure wPMRS rewrite rules. The component S is a distinguished non-terminal called the

"start symbol". The reduction relation for HORS, $\rightarrow$, is just that of wPMRS, noting that all redexes will necessarily be of the form $F\ \theta x_1\ \cdots\ \theta x_m$ since there are no pattern-matching arguments. We can associate with a recursion scheme $\mathcal{G}$ its language $\mathcal{L}(\mathcal{G})$ of terms in $T(\Sigma)$ that can be derived from the start symbol S by rewriting away all occurrences of non-terminals. More precisely, we make the following definition:

$$\mathcal{L}(\mathcal{G}) := \{\, t \mid S \rightarrow^* t,\ t \in T(\Sigma) \,\}$$

We define the *order* of a recursion scheme analogously with PMRS and wPMRS. Note that (as generators of finite ranked trees) order-0 recursion schemes are equivalent to regular tree grammars.

Trivial automata. Let Σ be as before. A *Büchi tree automaton with a trivial acceptance condition* (or simply, *trivial automaton*) is a quadruple $\mathcal{A} = \langle \Sigma, Q, \Delta, q_0 \rangle$ where Σ is as before, Q is a finite set of states, $q_0 \in Q$ is the initial state, and Δ, the transition relation, is a subset of $Q \times \Sigma \times Q^*$ such that if $(q, f, q_1 \cdots q_n) \in \Delta$ then $n = \mathsf{ar}(f)$. A Σ-labelled tree t is *accepted* by $\mathcal{A}$ if there is a Q-labelled tree r such that

(i) $\mathbf{dom}(t) = \mathbf{dom}(r)$,

(ii) for every $x \in \mathbf{dom}(r)$, $(r(x), t(x), r(x\,1) \cdots r(x\,m)) \in \Delta$ where $m = \mathsf{ar}(t(x))$.

The tree r is called a *run-tree* of $\mathcal{A}$ over t. We write $\mathcal{L}(\mathcal{A})$ for the set of Σ-labelled trees accepted by $\mathcal{A}$.

The PMRS Verification Problem. Given a deterministic PMRS $\mathcal{P} = \langle \Sigma, \mathcal{N}_\mathcal{P}, \mathcal{R}_\mathcal{P}, \mathit{Main} \rangle$, a (non-deterministic) order-0 recursion scheme $\mathcal{G} = \langle \Sigma, \mathcal{N}_\mathcal{G}, \mathcal{R}_\mathcal{G}, S \rangle$, and a Büchi tree automaton with a trivial acceptance condition $\mathcal{A} = \langle \Sigma, Q, \Delta, q_0 \rangle$, we write:

$$\models (\mathcal{P}, \mathcal{G}, \mathcal{A}) \quad \text{iff} \quad \forall t \in \mathcal{L}(\mathcal{G}) \cdot [\![\mathit{Main}\ t]\!]_\mathcal{P} \in \mathcal{L}(\mathcal{A})$$

The *PMRS Verification Problem* is to decide the truth of $\models(\mathcal{P}, \mathcal{G}, \mathcal{A})$.

3. Constructing an abstraction

In this section we will present an algorithm which, given an order-n deterministic PMRS $\mathcal{P}$ and an order-0 recursion scheme $\mathcal{G}$, constructs an order-n wPMRS $\widetilde{\mathcal{P}_\mathcal{G}}$ whose language of Σ-labelled trees is an over-approximation of the set of Σ-labelled trees reachable from $\mathcal{L}(\mathcal{G})$ under rewriting by $\mathcal{P}$.

At the heart of the algorithm is an analysis of the composite PMRS $\mathcal{P}_\mathcal{G} := \langle \Sigma, \mathcal{N}_\mathcal{G} \cup \mathcal{N}_\mathcal{P}, \mathcal{R}_\mathcal{G} \cup \mathcal{R}_\mathcal{P}, \mathit{Main} \rangle$. Since every term s reachable from $\mathcal{L}(\mathcal{G})$ under rewriting by $\mathcal{P}$ (i.e. $\mathit{Main}\ t \Rightarrow^*_\mathcal{P} s$, for some $t \in \mathcal{L}(\mathcal{G})$) is certainly reachable from S under rewriting by $\mathcal{P}_\mathcal{G}$ (i.e. $\mathit{Main}\ S \Rightarrow^*_{\mathcal{P}_\mathcal{G}} \mathit{Main}\ t \Rightarrow^*_{\mathcal{P}_\mathcal{G}} s$), it suffices to look only at the behaviours of $\mathcal{P}_\mathcal{G}$ in order to construct a safe abstraction of those of $\mathcal{P}$. We detail the nature of this analysis and its properties separately before showing how it underlies the construction of the approximating wPMRS $\widetilde{\mathcal{P}_\mathcal{G}}$.

Some nomenclature. A *simple term* is a subterm of the RHS of a $\mathcal{P}_\mathcal{G}$-rule or is the "starting term" $\mathit{Main}\ S$. A *compound term* has the shape $\xi\, t_1 \cdots t_m$ with $m \geq 0$, where the head symbol ξ is either a variable, or a terminal, or a non-terminal, and each t_i is simple. It follows from the definition that a simple term is compound, but the converse is not true.

3.1 Binding analysis

In a PMRS, the pattern matching rules use pattern matching both to determine control flow (by selecting which of a number of defining rules is used to reduce a redex) as well as to decompose compound data structure (by binding components to variables in the pattern that then occur on the RHS of the rule). However, the weak pattern matching mechanism in a wPMRS exhibits only the former capability: although patterns are matched, since there are no pattern-matching variables on the RHS of defining rules, data structures cannot be decomposed. Therefore, to build an effective abstraction of a PMRS requires some knowledge of the substitutions that can occur in redex/contractum pairs during PMRS reduction.

To this end, we define a *binding analysis*, which determines a (finitary) over-approximation Ξ to the set of variable-term bindings $\bigcup\{\, \theta \mid \mathit{Main}\ S \Rightarrow^* C[F\ \theta x_1\ \cdots\ \theta x_m\ \theta p] \Rightarrow C[\theta t] \,\}$ which occur in redex/contractum substitutions θ arising in $\mathcal{P}_\mathcal{G}$-reductions from '$\mathit{Main}\ S$'. The analysis is based on the observation that every such redex is either '$\mathit{Main}\ S$', or arises as an instance of a simple term. It proceeds by an iterative process in which bindings, by which instances of simple terms can be derived, give rise to redexes which in turn give rise, via contraction, to more bindings, until the desired set Ξ is reached in the limit.

Before we give the details of the analysis, let us make precise what it means for a set of bindings $\mathcal{S}$ to give rise to an instance of a term. Given such a set $\mathcal{S}$, we define the relation $s \preccurlyeq_\mathcal{S} t$, which is a subset of $T(\Sigma, \mathcal{N}, \mathcal{V}) \times T(\Sigma, \mathcal{N})$, inductively, by the system RS:

(R) $t \preccurlyeq_\mathcal{S} t$

(S) If $x \mapsto s \in \mathcal{S}$ and $C[s] \preccurlyeq_\mathcal{S} t$, then $C[x] \preccurlyeq_\mathcal{S} t$

where C ranges over one-hole contexts. We say that an instance of rule (S) is a *head-instance* just if the hole in $C[\,]$ occurs in head position.

Example 4. Let $\mathcal{S}_1 = \{x \mapsto y\ \mathtt{b},\ x \mapsto N,\ y \mapsto \mathtt{f}\ z,\ z \mapsto \mathtt{a}\}$. Then, using the system RS, it is possible to derive:

$$F\ x\ z \preccurlyeq_{\mathcal{S}_1} F\ (\mathtt{f\ a\ b})\ \mathtt{a} \qquad \text{and} \qquad F\ x\ z \preccurlyeq_{\mathcal{S}_1} F\ N\ \mathtt{a}$$

Note that the form of rule (S) does not constrain bindings to be used consistently within non-linear terms. Let $\mathcal{S}_2 = \{x \mapsto \mathtt{f}\ y\ z,\ y \mapsto z,\ y \mapsto \mathtt{a},\ z \mapsto \mathtt{b}\}$. Then we have, for example:

$$F\ x\ (G\ x) \preccurlyeq_{\mathcal{S}_2} F\ (\mathtt{f\ a\ b})\ (G\ (\mathtt{f\ b\ b}))$$

in which the binding $y \mapsto \mathtt{a}$ has been used in the derivation of the first argument of F whereas $y \mapsto z$ has been used in the derivation of the second argument.

To ensure that the analysis is computable, we cannot afford to work with instances of simple terms directly. We instead work with terms in which bindings have been applied only where strictly necessary in order to uncover new redexes. The construction of such terms is the purpose of the function head.

The head function. Given a set $\mathcal{S}$ of bindings, we define the *head function*, $\mathsf{head}_\mathcal{S} : T(\Sigma, \mathcal{N}, \mathcal{V}) \longrightarrow 2^{T(\Sigma, \mathcal{N}, \mathcal{V})}$ given by:

$$\mathsf{head}_\mathcal{S}(\xi\ t_1\ \cdots\ t_m) = \{\, \delta\ t_1\ \cdots\ t_m \mid \delta \in \mathsf{hs}_\mathcal{S}(\xi, \emptyset) \,\}$$

where $\mathsf{hs}_\mathcal{S}$ is an auxiliary function defined by the following:

$\mathsf{hs}_\mathcal{S}(k, X) = \{k\}$ (whenever $k \in \Sigma \cup \mathcal{N}$)

$\mathsf{hs}_\mathcal{S}(x, X) =$ if $x \in X$ then $\emptyset$ else

$$\{\, \delta\ t_1\ \cdots\ t_m \mid x \mapsto \zeta\ t_1\ \cdots\ t_m \in \mathcal{S},\ \delta \in \mathsf{hs}_\mathcal{S}(\zeta, X \cup \{x\}) \,\}$$

Thus $\mathsf{head}_\mathcal{S}(u)$ is the set of terms that are obtainable from u by iteratively replacing the head symbol—provided it is a variable—by a term bound to it in $\mathcal{S}$. The second argument of $\mathsf{hs}_\mathcal{S}$ disregards any cyclic chain of bindings. For example, let $\mathcal{S} = \{\, x \mapsto y, y \mapsto x \,\}$, then: $\mathsf{head}_\mathcal{S}(x) = \mathsf{hs}_\mathcal{S}(x, \emptyset) = \mathsf{hs}_\mathcal{S}(y, \{\, x \,\}) = \mathsf{hs}_\mathcal{S}(x, \{\, x, y \,\}) = \emptyset$

Example 5. Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be as in Example 4. Then:

$$\mathsf{head}_{\mathcal{S}_1}(x\ \mathtt{c}) = \{\, N\ \mathtt{c},\ \mathtt{f}\ z\ \mathtt{b}\ \mathtt{c} \,\}$$
$$\mathsf{head}_{\mathcal{S}_2}(x\ \mathtt{c}) = \{\mathtt{f}\ y\ z\ \mathtt{c}\} \quad \mathsf{head}_{\mathcal{S}_2}(F\ x\ (G\ x)) = \{F\ x\ (G\ x)\}$$

Notice that since head performs variable-substitutions according to bindings from $\mathcal{S}$, its behaviour is consistent with a strategy

for constructing initial prefixes of derivations in the system RS. Each use of the recursive clause of $\mathsf{hs}_{\mathcal{S}}$ corresponds to a head-instance of rule (S). A consequence of this relationship is made precise by the following lemma.

Lemma 1. *If $u \preccurlyeq_{\mathcal{S}} \xi\ v_1\ \cdots\ v_m$ then there is a compound term $\xi\ u_1\ \cdots\ u_m \in \mathsf{head}_{\mathcal{S}}(u)$ and, for all $1 \leq i \leq m$, $u_i \preccurlyeq_{\mathcal{S}} v_i$.*

One final property to note about head is that, whenever its argument is compound and all the variables in $\mathcal{S}$ are bound to simple terms, the terms in (sets in) its image are all compound. This is due to the fact that, in this case, the action of the head function is to construct new, compound terms by prepending old, simple terms into head position. This limited behaviour of the head-function will contribute towards guaranteeing the termination of the analysis.

Lemma 2. *We say that a set of bindings $\mathcal{S}$ is image-simple just if every term in the image of $\mathcal{S}$ is simple. Suppose $\mathcal{S}$ is image-simple. If u is compound, then every term in $\mathsf{head}_{\mathcal{S}}(u)$ is compound.*

The goal of the analysis is to discover the possible redexes $F\theta x_1\ \cdots\ \theta x_m\ \theta p$ that occur during reduction sequences of $\mathcal{P}_{\mathcal{G}}$ starting from $Main\ S$. The head function $\mathsf{head}_{\mathcal{S}}(u)$ is able to determine, in a way that is computable, when an F-redex is an instance (according to $\mathcal{S}$) of a simple term u. In this case, according to Lemma 1, a term of the shape $F\ t_1\ \cdots\ t_m\ s$ is an element of $\mathsf{head}_{\mathcal{S}}(u)$. However, to know which defining rule of F is triggered, it is necessary to find out which patterns are matched by residuals of instances of s.

The approximate reduction. To this end, we introduce a new notion of reduction $\triangleright_{\mathcal{S}} \subseteq T(\Sigma,\mathcal{N},\mathcal{V}) \times T(\Sigma,\mathcal{N},\mathcal{V})$ parametrised by a set of bindings $\mathcal{S}$. This reduction approximates the usual PMRS reduction by performing redex/contractum substitutions only where absolutely necessary and only when the relevant bindings are contained in $\mathcal{S}$. A *$\triangleright_{\mathcal{S}}$-redex* is a term of the form $F\ \theta x_1\ \cdots\ \theta x_m\ \theta p$ whenever there is a $\mathcal{P}_{\mathcal{G}_0}$-rule of the form $F\ x_1\ \cdots\ x_m\ p \longrightarrow t$ and θ is a substitution (not necessarily closed). The contractum of the redex is t, no substitution is performed upon contraction.

We define the one step reduction $\triangleright_{\mathcal{S}}$ by the following rules. Let C range over one-hole contexts.

$$\frac{(s,t) \text{ a } \triangleright\text{-redex/contractum pair}}{C[s] \triangleright_{\mathcal{S}} C[t]} \qquad \frac{t \in \mathsf{head}_{\mathcal{S}}(x\,t_1\ \cdots\ t_m)}{C[x\ t_1\ \cdots\ t_m] \triangleright_{\mathcal{S}} C[t]}$$

As is standard, we write $\triangleright_{\mathcal{S}}^*$ to mean the reflexive, transitive closure of $\triangleright_{\mathcal{S}}$, and $\triangleright_{\mathcal{S}}^n$ to mean a n-long chain of $\triangleright_{\mathcal{S}}$.

Example 6. Consider the composite PMRS $\mathcal{P}_{\mathcal{G}}$ constructed from the PMRS and grammar given in Example 2 and let $\mathcal{S}$ contain the bindings $p \mapsto Nz$ and $x \mapsto N$. Then the following:

$$p\ x \triangleright_{\mathcal{S}} Nz\ x \triangleright_{\mathcal{S}} Nz\ N\ \triangleright_{\mathcal{S}} Nz\ (\mathtt{s}\ N) \triangleright_{\mathcal{S}} \mathtt{true}$$

is a $\triangleright_{\mathcal{S}}$-reduction. Observe how, as demonstrated by the third step, approximate reduction is accurate for order-0 $\mathcal{G}$-rules.

Given a substitution θ and a pattern p, we say that a $\triangleright_{\mathcal{S}}$-reduction $s \triangleright_{\mathcal{S}}^i \theta p$ is *minimal* just if it is *not* the case that there exist $j < i$ and substitution θ' such that $s \triangleright_{\mathcal{S}}^j \theta'\ p$. Consider the two rules defining $\triangleright_{\mathcal{S}}$-reduction. In the RHS of the conclusion of each rule is the term t. In both cases, assuming $\mathcal{S}$ is image simple, t is a compound term. Since there are only finitely many such terms t and since there are only finitely many patterns (drawn from the PMRS) p, the problem of finding such *minimal* reductions is computable.

Lemma 3. *Assume $\mathcal{S}$ is image-simple. Given a compound term s and a pattern p drawn from the defining rules of $\mathcal{P}_{\mathcal{G}}$, the problem of finding a substitution θ and a minimal reduction $s \triangleright_{\mathcal{S}}^* \theta p$ is computable.*

The fixpoint construction. Let $\mathcal{S}$ be a set of bindings. We define $\mathcal{F}(\mathcal{S})$ as the least set X of bindings that contains $\mathcal{S}$ and is closed under *Rule* **C**: if

(i) u is simple term of base type,

(ii) $F\ t_1\ \cdots\ t_m\ s \in \mathsf{head}_{\mathcal{S}}(u)$,

(iii) $F\ x_1\ \cdots\ x_m\ p \longrightarrow t$ is a $\mathcal{P}_{\mathcal{G}_0}$-rule,

(iv) there is a minimal reduction $s \triangleright_{\mathcal{S}}^* \theta\ p$

then $\theta \cup \{\, x_i \mapsto t_i \mid 1 \leq i \leq m \,\} \subseteq X$.

Thus $\mathcal{F} : 2^{\mathcal{V}\times T(\Sigma,\mathcal{N},\mathcal{V})} \longrightarrow 2^{\mathcal{V}\times T(\Sigma,\mathcal{N},\mathcal{V})}$ is, by construction, a monotone (endo)function on the complete lattice $2^{\mathcal{V}\times T(\Sigma,\mathcal{N},\mathcal{V})}$ ordered by subset-inclusion. By the Tarski-Knaster Fixpoint Theorem, the least fixpoint of $\mathcal{F}$, which we shall denote Ξ, exists, and is constructable as the supremum of the chain

$$\emptyset \subseteq \mathcal{F}\,\emptyset \subseteq \mathcal{F}(\mathcal{F}\,\emptyset) \subseteq \mathcal{F}(\mathcal{F}(\mathcal{F}\,\emptyset)) \subseteq \cdots$$

Example 7. Consider again the composite PMRS $\mathcal{P}_{\mathcal{G}}$ composed from the PMRS $\mathcal{P}$ and tree grammar $\mathcal{G}$ given in Example 2. We shall apply the fixpoint construction to this structure.

Initially, the only fruitful choice of simple term is the "starting term" $Main\ S$ which otherwise trivially satisfies the premises of Rule **C** and yields the single binding $m \mapsto S$. Subsequently, taking $u = Filter\ Nz\ m$ matches both the defining rules for $Filter$ after approximate-reductions of:

$$m \triangleright_{\{m\mapsto S\}}^* \mathtt{nil} \qquad \text{and} \qquad m \triangleright_{\{m\mapsto S\}}^* \mathtt{cons}\ N\ ListN$$

respectively. This choice adds the bindings $p \mapsto Nz$, $x \mapsto N$ and $xs \mapsto ListN$. Examining the term $p\ x$ in the RHS of the second defining rule for $Filter$ then gives $n \mapsto N$. Finally, taking u as the entire RHS of the second defining rule for $Filter$ and approximate-reducing $p\ x$ as in Example 6 gives bindings $a \mapsto \mathtt{cons}\ x\ (Filter\ p\ xs)$ and $b \mapsto Filter\ p\ xs$. In this case, no other choices of simple term yield any new bindings, so the fixpoint Ξ is obtained as:

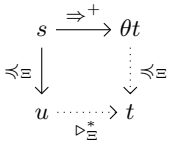
$$m \mapsto S, \quad p \mapsto Nz, \quad x \mapsto N, \quad xs \mapsto ListN$$
$$n \mapsto N, \quad a \mapsto \mathtt{cons}\ x\ (Filter\ p\ xs), \quad b \mapsto Filter\ p\ xs$$

Though the complete lattice $2^{\mathcal{V}\times T(\Sigma,\mathcal{N},\mathcal{V})}$ is infinite, the least fixpoint Ξ is finitely constructable (i.e. the closure ordinal of $\mathcal{F}$ is finite); it is in fact a finite set. Observe that, in Example 7, the form of every binding in the fixpoint is $v \mapsto t$ in which t is a simple term. This is the key to showing the convergence of the analysis. Since every term $F\ t_1\ \cdots\ t_m\ s \in \mathsf{head}_{\mathcal{S}}(u)$ is compound (whenever $\mathcal{S}$ is image-simple and u is compound) so every binding $x_i \rightarrow t_i$ is image-simple. Since, whenever $\mathcal{S}$ is image-simple, every $\triangleright_{\mathcal{S}}$-contractum is compound, so the bindings due to θp are image-simple. Since there are only finitely many simple terms, termination follows.

Theorem 1 (Termination)**.** *The least fixpoint of $\mathcal{F}$, Ξ, is a finite set.*

To see that this finite set of bindings Ξ is sufficient to describe all the all the substitutions that occur during redex contractions in reduction sequences of $\mathcal{P}_{\mathcal{G}}$ starting from $Main\ S$, one should first notice that the approximate reduction, when instantiated with the fixpoint, acts on simple terms in a way which is consistent with the way PMRS reduction acts on their instances in a trivial context.

Lemma 4. *Assume θt is a contractum and u is a simple term. If $s \Rightarrow^+ \theta t$ and $u \preccurlyeq_\Xi s$, then $u \triangleright_\Xi^* t$ and $t \preccurlyeq_\Xi \theta t$.*



To lift this fact to the level of arbitrary reduction sequences starting from $Main\ S$, it is enough to observe that any redex in such a sequence (apart from the first), can be seen either to be itself a simple term or to arise as a subterm of some previous contractum, regardless of the context in which the redex occurs. As a consequence of Lemma 4, the variable-term bindings necessary to derive the redex as an instance of the corresponding simple term will already be contained in the fixpoint. Hence, if the reduction sequence reaches any contractum, the fixpoint will contain the bindings necessary to reconstruct the substitution associated with the contraction.

Lemma 5. *Assume θt is a contractum. If $Main\ S \Rightarrow^+ C[\theta t]$ is a $\mathcal{P}_\mathcal{G}$-reduction sequence then $t \preccurlyeq_\Xi \theta t$.*

3.2 Construction of the over-approximating wPMRS

We are now ready to define the wPMRS which is an abstraction of the composite PMRS $\mathcal{P}_\mathcal{G} = \langle \Sigma, \mathcal{N}, \mathcal{R}, Main \rangle$. Let Ξ be the fixpoint set of bindings and let $\mathcal{N}_\mathcal{V} = \{ V_x \mid x \in \mathcal{V} \}$ and $\mathcal{N}_\Sigma = \{ K_\mathtt{a} \mid \mathtt{a} \in \Sigma \}$ be two sets of fresh non-terminal symbols which we call *pattern-symbols* and *accounting-symbols* respectively. We define the approximating wPMRS:

$$\widetilde{\mathcal{P}_\mathcal{G}} := \langle \Sigma,\ \mathcal{N} \cup \mathcal{N}_\mathcal{V} \cup \mathcal{N}_\Sigma,\ \mathcal{R}',\ Main \rangle$$

where $\mathcal{R}'$ consists of the following three kinds of rules:

I. *Weak pattern-matching rules*. For each (pure or pattern-matching) $\mathcal{P}_\mathcal{G}$-rule $F\, x_1 \cdots x_m\, p \longrightarrow t$, $\mathcal{R}'$ contains the following rule:

$$F\, x_1 \cdots x_m\, p \longrightarrow t^\dagger$$

II. *Instantiation rules*. For each binding $x \mapsto t$ in Ξ where $\mathsf{FV}(t^\dagger) = \{ x_1, \cdots, x_l \}$, $\mathcal{R}'$ contains the following rule:

$$V_x\ z_1 \cdots z_{\mathsf{ar}(x)} \longrightarrow (t^\dagger[V_{x_1}/x_1] \cdots [V_{x_l}/x_l])\ z_1 \cdots z_{\mathsf{ar}(x)}$$

where each z_i is a fresh variable of the appropriate types.

III. *Accounting rules*. For each terminal symbol $\mathtt{a} : b_1 \to \cdots \to b_n \to o$ in Σ, $\mathcal{R}'$ contains the following rule:

$$K_\mathtt{a}\ z_1\ \cdots\ z_n \longrightarrow \mathtt{a}\ z_1\ \cdots\ z_n$$

where each z_i is a fresh variable of type b_i.

where we have written $t^\dagger$ to denote the term t in which every occurrence of a pattern matching variable $y \in \mathsf{FV}(p)$ has been replaced by the corresponding pattern-symbol V_y and every occurrence of a terminal symbol $\mathtt{a}$ has been replaced by the corresponding accounting-symbol $K_\mathtt{a}$.

Example 8. Consider the following order-2 PMRS, whose defining rules are given by:

$$\begin{array}{l} Main\ m \longrightarrow\ Map2\ KZero\ KOne\ m \\ \\ Map2\ \varphi\ \psi\ \mathtt{nil} \longrightarrow\ \mathtt{nil} \\ Map2\ \varphi\ \psi\ (\mathtt{cons}\ x\ xs) \longrightarrow\ \mathtt{cons}\ (\varphi\ x)\ (Map2\ \psi\ \varphi\ xs) \\ \\ KZero\ x_1 \longrightarrow\ \mathtt{0} \\ KOne\ x_2 \longrightarrow\ \mathtt{1} \end{array}$$

and input grammar $\mathcal{G}$ consisting of two rules:

$$S \longrightarrow\ \mathtt{nil} \mid \mathtt{cons}\ \mathtt{0}\ S$$

The function $Map2$ behaves like the standard Map function, except that it swaps the first two function arguments as it filters through the successive elements of the list argument. The reachable constructor terms are finite lists that are prefixes of $[0\ 1\ 0\ 1\ 0\ 1\ \cdots]$.

After applying the fixpoint construction to this example, the set of bindings Ξ consists of the following:

$$\begin{array}{llllllllll} m & \mapsto & S & \varphi & \mapsto & KZero & \varphi & \mapsto & \psi \\ x & \mapsto & \mathtt{0} & \psi & \mapsto & KOne & \psi & \mapsto & \varphi \\ x_1 & \mapsto & x & x_2 & \mapsto & x & xs & \mapsto & S \end{array}$$

and hence the approximating wPMRS $\widetilde{\mathcal{P}_\mathcal{G}}$ is as follows.

$$\begin{array}{l} Main\ m \longrightarrow\ Map2\ KZero\ KOne\ M \\ \\ Map2\ \varphi\ \psi\ \mathtt{nil} \longrightarrow\ Nil \\ Map2\ \varphi\ \psi\ (\mathtt{cons}\ x\ xs) \longrightarrow\ Cons\ (\varphi\ X)\ (Map2\ \psi\ \varphi\ XS) \\ \\ KZero\ x_1 \longrightarrow\ Zero \\ KOne\ x_2 \longrightarrow\ One \\ \\ M \longrightarrow\ S \\ X \longrightarrow\ Zero \\ XS \longrightarrow\ S \\ S \longrightarrow\ Nil \mid Cons\ Zero\ S \\ \\ Zero \longrightarrow\ \mathtt{0} \\ One \longrightarrow\ \mathtt{1} \\ Nil \longrightarrow\ \mathtt{nil} \\ \\ Cons\ v_1\ v_2 \longrightarrow\ \mathtt{cons}\ v_1\ v_2 \end{array}$$

Since φ, ψ, x_1 and x_2 are not pattern-matched variables, the rules for V_φ, V_ψ, V_{x_1} and V_{x_2} are, in this case, never used and so play no part in the approximation process: they have been omitted. It is easy to see that the constructor terms in $\mathcal{L}(\widetilde{\mathcal{P}_\mathcal{G}}, Main\ S)$ are exactly the finite prefixes of $[0\ 1\ 0\ 1\ \cdots]$ i.e. the approximation is exact in this case.

Given any $\mathcal{P}_\mathcal{G}$-reduction $Main\ S \Rightarrow^+ t$, the reduction can be faithfully simulated in the abstraction $\widetilde{\mathcal{P}_\mathcal{G}}$ using the weak pattern-matching rules and the instantiation rules. Whenever the $\mathcal{P}_\mathcal{G}$-reduction contracts a $\mathcal{P}$-rule which binds data θy to a pattern matching variable y, the simulation can contract the corresponding redex using a weak pattern-matching rule and, by Lemma 5, can then reconstruct the bound data θy from V_y using the instantiation rules.

Theorem 2 (Soundness)**.** *Let the composite PMRS $\mathcal{P}_\mathcal{G}$ and the approximating wPMRS $\widetilde{\mathcal{P}_\mathcal{G}}$ be as before. Then $\mathcal{L}(\mathcal{P}_\mathcal{G}, Main\ S) \subseteq \mathcal{L}(\widetilde{\mathcal{P}_\mathcal{G}}, Main\ S)$.*

The third class of rules is not essential to the achieving soundness. The purpose of the accounting rules is to enforce a strict correspondence between the length of a $\widetilde{\mathcal{P}_\mathcal{G}}$ reduction sequence and the maximum size of any constructor term created within it. This eases the justification of the semi-completeness property of refinement in Section 5.

4. Model checking by type inference

In this section, we exhibit an algorithm to decide the *wPMRS Model Checking Problem*: given a non-deterministic wPMRS $\mathcal{W} = \langle \Sigma, \mathcal{N}, \mathcal{R}, Main \rangle$ in which $Main : b \to o$, a closed term $t : b$ and a trivial automaton $\mathcal{A}$, is $\mathcal{L}(\mathcal{W}, Main\ t) \subseteq \mathcal{L}(\mathcal{A})$? Following work by Kobayashi [6] and Kobayashi and Ong [8], we characterise the model checking problem as a type inference problem in a particular, finitary intersection type system induced by the automaton.

Eliminating non-determinism. The first step we take is to simplify the problem at hand by eliminating the non-determinism in $\mathcal{W}$. To this end we construct a new wPMRS $\mathcal{W}^{\#}$ in which multiple defining rules for a given non-terminal are collapsed using a family $\mathcal{B} := \{ \mathtt{br}_b \mid b \in \mathbb{B} \}$ of "non-deterministic choice" terminal symbols $\mathtt{br}_b$ of type $b \to b \to b$. We define:

$$\mathcal{W}^{\#} := \langle \Sigma \cup \mathcal{B},\, \mathcal{N},\, \{ l \longrightarrow \mathtt{BR}(l) \mid \exists r \cdot l \longrightarrow r \in \mathcal{R} \},\, \mathit{Main} \rangle$$

in which, by way of a short-hand, we define:

$$\mathtt{BR}(F\ t_1\ \cdots\ t_n) := \mathtt{br}_b\ r_1\ (\mathtt{br}_b\ r_2\ (\cdots(\mathtt{br}_b\ r_{m-1}\ r_m)\cdots))$$

where $\{r_1, \ldots, r_m\} = \{ r \mid F\ t_1\ \cdots\ t_n \longrightarrow r \in \mathcal{R} \}$ and the type of F is of the form $\tau_1 \to\ \cdots\ \to \tau_n \to b$. We must modify the automaton $\mathcal{A}$ accordingly, so we define:

$$\mathcal{A}^{\#} := \langle \Sigma \cup \mathcal{B},\, Q,\, \Delta \cup \{ (q,\, \mathtt{br_b},\, q\,q \mid q \in Q,\, b \in \mathbb{B} \},\, q_0 \rangle$$

Lemma 6. *For all terms t of base-type:*

$$\mathcal{L}(\mathcal{W}, \mathit{Main}\ t) \subseteq \mathcal{L}(\mathcal{A}) \qquad \textit{iff} \quad [\![\mathit{Main}\ t]\!]_{\mathcal{W}^{\#}} \in \mathcal{L}(\mathcal{A}^{\#})$$

Model checking as type inference. We first introduce recursion schemes with weak definition-by-cases, which is a term rewriting system similar to (in fact, equi-expressive with) wPMRS; the difference is that (weak) matching is explicitly provided by a case construct. Assume for each base type b, an exhaustive and non-overlapping family of patterns $P_b = \{ p_1, \cdots, p_k \}$. A *recursion scheme with weak definition-by-cases* (wRSC) is a quadruple $\mathcal{G} = \langle \Sigma, \mathcal{N}, \mathcal{R}, S \rangle$ where $\Sigma, \mathcal{N}$, and S are as usual, and $\mathcal{R}$ is a set of (pure) rules of the form

$$F\, x_1 \cdots x_m \longrightarrow t$$

We write $\mathbf{rhs}(F) = \lambda x_1 \cdots x_m.t$. The set of applicative terms is defined as before, except that it is augmented by a definition-by-cases construct $\mathsf{case}_b(t; t_1, \cdots, t_k)$ with typing rule:

$$\frac{\Gamma \vdash t : b \qquad \Gamma \vdash t_i, o \text{ (for } 1 \le i \le k)}{\Gamma \vdash \mathsf{case}_b(t; t_1, \cdots, t_k) : o}$$

We say that $\mathcal{G}$ is deterministic just if there is one rule for each $F \in \mathcal{N}$. There are two kinds of redexes:

(i) $F\, s_1 \cdots s_m$ which contracts to $t[s_1/x_1] \cdots [s_m/x_m]$ for each rule $F\, x_1 \cdots x_m \longrightarrow t$ in $\mathcal{R}$

(ii) $\mathsf{case}_b(t; t_1, \cdots, t_k)$ which contracts to t_i, provided t of base type b matches pattern $p_i \in P_b = \{ p_1, \cdots, p_k \}$.

We define evaluation contexts E as follows

$$E \ ::= \ [\,] \mid f\, t_1 \cdots t_{i-1}\, E\, t_{i+1} \cdots t_{\mathsf{ar}(f)}$$

and write $\to$ for the one-step reduction relation $E[\Delta] \to E[\Delta]$ where (Δ, Δ) ranges over redex/contractum pairs and E over evaluation contexts. Assuming $\mathcal{G}$ is deterministic, we define the $\Sigma^{\perp}$-labelled tree generated by $\mathcal{G}$ by infinitary rewriting from S as $[\![\mathcal{G}]\!] := \{ t^{\perp} \mid S \to^* t \}$.

Lemma 7. *Deterministic wPMRS and deterministic wRSC are equi-expressive as generators of Σ-labelled trees.*

We present an intersection type system for characterising the model checking problem. The *intersection types* of the system are given by the grammar:

$$\sigma, \tau \quad ::= \quad q \ \mid\ p \ \mid\ \bigwedge_{i=1}^{m} \tau_i \to \tau$$

where $q \in Q$ and p is one of the finitely many patterns associated with a definition by cases in the scheme $\mathcal{G}$. Judgements of the type system are sequents of the form $\Gamma \vdash t : \tau$, in which Γ is simply a set of type bindings $\xi : \sigma$ where $\xi \in \mathcal{N} \cup \mathcal{V}$. The defining rules of the system are as follows:

$$\frac{}{\Gamma, x : \tau \vdash x : \tau} \qquad (\textsc{Var})$$

$$\frac{(q,\, f,\, q_1 \cdots q_n) \in \Delta_{\mathcal{A}^{\#}}}{\Gamma \vdash f : q_1 \to \cdots \to q_n \to q} \qquad (\textsc{Term})$$

$$\frac{\exists \theta \cdot s\ p_1\ \cdots\ p_n = \theta\, p}{\Gamma \vdash s : p_1 \to \cdots \to p_n \to p} \qquad (\textsc{Match})$$

$$\frac{\Gamma \vdash t : p_i \qquad \Gamma \vdash t_i : \tau}{\Gamma \vdash \mathsf{case}_b(t;\ t_1, \cdots, t_i, \cdots, t_n) : \tau} \qquad (\textsc{Case})$$

$$\frac{\Gamma \vdash s : \bigwedge_{i=1}^{n} \tau_i \to \tau \qquad \Gamma \vdash t : \tau_i \quad (\text{for each } 1 \le i \le n)}{\Gamma \vdash s\ t : \tau} \qquad (\textsc{App})$$

$$\frac{\Gamma, x : \tau_1, \cdots, x : \tau_n \vdash t : \tau}{\Gamma \vdash \lambda x.t : \bigwedge_{i=1}^{n} \tau_i \to \tau} \qquad (\textsc{Abs})$$

Note that we have the following derived rule from (Match): if a term s (of the appropriate type) matches the pattern p, then $\Gamma \vdash s : p$.

We write $\vdash_{\mathcal{A}} \mathcal{G} : \Gamma$ if $\Gamma \ \vdash\ \mathbf{rhs}(F) : \tau$ is provable for every $F : \tau \in \Gamma$. A wRSC is *well-typed*, written $\vdash_{\mathcal{A}} \mathcal{G}$, just if there exists Γ such that (i) $\vdash_{\mathcal{A}} \mathcal{G} : \Gamma$, (ii) $S : q_0 \in \Gamma$, (iii) for each $F : \tau \in \Gamma$, $\tau :: \kappa$, where $F : \kappa \in \mathcal{N}$, meaning that τ is an intersection type *compatible* with type κ (as assigned to F by the wRSC), which is defined by: (i) $q :: o$, (ii) $p :: b$ for each $p \in P_b$, (iii) $\bigwedge_{i=1}^{k} \tau_i \to \tau :: \kappa' \to \kappa$ if $\tau :: \kappa$ and for each $1 \le i \le k$, $\tau_i :: \kappa'$.

Theorem 3. *Let $\mathcal{A}$ be a trivial automaton, and $\mathcal{G}$ be wRSC. Then $\vdash_{\mathcal{A}} \mathcal{G}$ if and only if $[\![\mathcal{G}]\!] \in \mathcal{L}(\mathcal{A})$.*

The proof is omitted as it is very similar to the proof of the soundness and completeness theorems in [6].

Corollary 1. *The wPMRS model checking problem is decidable.*

Proof. This follows from Lemma 6, Lemma 7 and Theorem 3, and the decidability of typability $\vdash_{\mathcal{A}} \mathcal{G}$. The latter follows from the fact that for each non-terminal, there are only finitely many candidate intersection types compatible with a given type. □

5. Abstraction refinement

When the model checking stage reports a counterexample in the form of an error trace, the trace may be "feasible", that is, it corresponds to a concrete reduction sequence in the original PMRS $\mathcal{P}$, or it may be "spurious": an artifact of the abstraction process. In the case the counterexample is spurious, we will want to ignore it and perform the process again, but in a new setting in which we are guaranteed never again to encounter this unwanted trace. To achieve this we restart the cycle from a modified PMRS $\mathcal{P}'$, which has had some of its defining rules unfolded, so as to reduce the amount of non-determinism in the corresponding wPMRS abstraction.

5.1 Counterexamples and feasibility.

When the model-checker reports a violation of the property, a counterexample error trace is returned. This error trace is a reduction sequence in the abstract wPMRS $\widetilde{\mathcal{P}_\mathcal{G}}$. Since $\widetilde{\mathcal{P}_\mathcal{G}}$ is not completely faithful to the PMRS $\mathcal{P}$, it is necessary to determine whether such a counterexample trace corresponds to a reduction sequence in $\mathcal{P}$ which itself witnesses the violation or whether it is an artifact of the abstraction.

Anatomy of a counterexample. It is useful to highlight two important features of any given counterexample trace, namely, (i) the shape of the last term in the reduction sequence and (ii) the "type" of each constituent reduction .

Any counterexample trace must end in a term t which witnesses the violation of the property φ. Since the property is a collection of (possibly infinite) Σ-labelled trees, the witnessing term can be seen to be of the form θq where q is a pattern which does not match any prefix of a tree $t \in \varphi$. We say that the pattern q which witnesses the violation of the property is the *error witness*.

In any $\widetilde{\mathcal{P}_\mathcal{G}}$ reduction sequence, each reduction $u \to v$ can be classified into one of two kinds based on the head symbol occurring in the redex. In case we want to emphasise that the head symbol is a non-terminal belonging to $\mathcal{P}$ we say the contraction of this redex is an *abstract $\mathcal{P}$-reduction* and write $u \to_\mathcal{P} v$. Otherwise the head symbol is either a pattern-symbol, an accounting-symbol or it belongs to $\mathcal{G}$. In this case we say that the head symbol in question is a *live-symbol* and that the contraction of this redex is an *abstract $\widetilde{\mathcal{P}_\mathcal{G}}$-reduction*; we write $u \to_{\widetilde{\mathcal{P}_\mathcal{G}}} v$.

Example 9. Consider the following abstract error trace which is derived from the abstraction $\widetilde{\mathcal{P}_\mathcal{G}}$ of the PMRS $\mathcal{P}$ and grammar $\mathcal{G}$ given in Example 2:

$Main\ S$
$\to\ \ Filter\ Nz\ M$
$\to^*\ Filter\ Nz\ (\texttt{cons}\ N\ ListN)$
$\to\ \ If\ (Cons\ X\ (Filter\ Nz\ XS))\ (Filter\ Nz\ XS)\ (Nz\ X)$
$\to^*\ If\ (Cons\ X\ (Filter\ Nz\ XS))\ (Filter\ Nz\ XS)\ (Nz\ (\texttt{s}\ N))$
$\to\ \ If\ (Cons\ X\ (Filter\ Nz\ XS))\ (Filter\ Nz\ XS)\ True$
$\to^*\ \ Cons\ X\ (Filter\ Nz\ XS)$
$\to\ \ \texttt{cons}\ X\ (Filter\ Nz\ XS)$
$\to^*\ \texttt{cons}\ \texttt{z}\ (Filter\ Nz\ XS)$

which violates the property since it is the start of a list that contains a zero. The *error-witness* for this trace is $\texttt{cons}\ \texttt{z}\ v$ (for some variable v). The first reduction is an *abstract $\mathcal{P}$-reduction*, as is the reduction written over lines 3 and 4 and that of lines 5 and 6. All the other reductions in the sequence are *abstract $\widetilde{\mathcal{P}_\mathcal{G}}$-reductions*.

The trace in the above example is spurious since there are no reduction sequences of the PMRS $\mathcal{P}$ (starting from terms in $\mathcal{L}(\mathcal{G})$) from Example 2 which result in a list headed by a zero. Intuitively, we can see that this trace is infeasible because the non-determinism introduced by the abstraction has been resolved in an inconsistent way during the sequence. The data bound by the pattern match for $Filter$, which is given as N (i.e. some number) has been resolved on the one hand (line 5) to a non-zero number and on the other hand (line 9) to zero.

In the following, we define a process of labelling of the counterexample trace that will reveal information about the resolution of non-determinism that has been introduced as a consequence of the abstraction. The information that is exposed will allow us to see whether or not this abstract trace in $\widetilde{\mathcal{P}_\mathcal{G}}$ has any corresponding trace in $\mathcal{P}$ starting form $\mathcal{I}$, that is, whether the trace is feasible.

Labelling. The labelling procedure, $\mathsf{labelSeq}$, keeps track of how non-determinism is resolved in an abstract reduction sequence by annotating each live-symbol X with a set of (possibly open) terms, which represent all the closed terms to which it reduces. When the terms are given by the set l, we write the annotated term X^l and we identify an unlabelled live-symbol X with $X^\emptyset$. Given a term t which may include labelled subterms, we define the *resolution* of t, which is a set of terms $\bar{t}$, defined as follows:

$$\bar{t} = \begin{cases} \{\mathtt{a}\} & \text{when } t = \mathtt{a} \in \Sigma \\ \{F\} & \text{when } t = F \text{ is not a live-symbol} \\ \hat{l} & \text{when } t = F^l \text{ is a live-symbol} \\ \{\, u\ v \mid u \in \overline{t_0},\, v \in \overline{t_1}\,\} & \text{when } t = t_0\ t_1 \end{cases}$$

where $\hat{l}$ denotes the set l when l is non-empty and $\{z\}$ for some fresh variable z otherwise. If any pattern-symbol reduces to two incompatible terms or to a term which is inconsistent with the term that it represents in the matching, then the procedure will detect a conflict and record it in the set $\mathsf{Failures}$.

$\mathsf{labelSeq}(Main\ S)$

If S is labelled by l and there is a term $t \in \mathcal{L}(\mathcal{G})$ which is an instance of $\mathsf{mgci}(l)$ then do nothing else add $(Main, l)$ to $\mathsf{Failures}$.

$\mathsf{labelSeq}(Main\ S \to^* u \to v)$

1. Analyse the reduction $u \to v$:
   $C[F\ \theta x_1\ \cdots\ \theta x_m] \to_{\widetilde{\mathcal{P}_\mathcal{G}}} C[t\ \theta x_1\ \cdots\ \theta x_m]$:
   Label the head symbol F by $\bar{t}$.
   $C[F\ \theta x_1\ \cdots\ \theta x_m\ \theta p] \to_\mathcal{P} C[\theta t^\dagger]$:
   For each $y \in \mathsf{FV}(p)$, let $\{V_y^{l_1}, \ldots, V_y^{l_k}\}$ be the labelled pattern-symbols in v created by the contraction. Perform $\mathsf{labelTm}(\theta y)(\bigcup\{l_1, \ldots, l_k\})$ on the corresponding occurrence of θy in θp. If $\mathsf{labelTm}$ fails, then add $(F, \bigcup\{l_1, \ldots, l_k\})$ to $\mathsf{Failures}$.
2. For each occurrence of an unlabelled live-symbol N in u, let $\{N_1^{l_1}, \ldots, N_k^{l_k}\}$ be the set of labelled descendants in v. Label this occurrence of N with $\bigcup\{l_1, \ldots, l_k\}$.
3. Perform $\mathsf{labelSeq}(Main\ S \to^* u)$.

where the procedure $\mathsf{labelTm}$, which is designed to resolve the data bound in a pattern match and the data created by the abstraction, is given by:

$\mathsf{labelTm}(t)(\{s_1, \ldots, s_k\})$

Analyse the form of t:

$t = \mathtt{a}$**:** If $\forall i$ $\mathtt{a}$ matches s_i then do nothing else fail.

$t = F$**:** If F is not a live-symbol and $\forall i$ F matches s_i then do nothing else, if F is a live-symbol and $w = \mathsf{mgci}(\{s_1, \ldots, s_k\})$ exists then label F by $\{w\}$ else fail.

$t = t_0\ t_1$**:** If $\forall i$ either s_i is a variable or $s_i = s_{i_0}\ s_{i_1}$ then (let $s_{j_0} = z_0$ and $s_{j_1} = z_1$ for fresh z_1, z_2 whenever s_j is a variable) and perform $\mathsf{labelTm}(t_0)(\{s_{1_0}, \ldots, s_{k_0}\})$ and perform $\mathsf{labelTm}(t_1)(\{s_{1_1}, \ldots, s_{k_1}\})$ else fail.

where $\mathsf{mgci}(l)$ denotes the most general common instance (MGCI) of the set of terms l (regarding a single fresh variable as the MGCI of the empty set)[1]. We call a counterexample trace that has been labelled by $\mathsf{labelSeq}$ a *labelled trace*.

[1] For the purposes of calculating MGCIs, terms are considered as first order entities constructed from atomic constants and a single (silent) application operator.

Example 10. Consider again the abstract reduction sequence in Example 9, after performing $\mathsf{labelSeq}$ the following labelled trace is produced (the set bracket notation has been elided since all labels are singleton sets):

$Main\ S^{\mathtt{cons}\ v_1\ v_2}$
$\rightarrow\ \ Filter\ Nz\ M^{\mathtt{cons}\ v_1\ v_2}$
$\rightarrow^*\ Filter\ Nz\ (\mathtt{cons}\ N\ ListN^{v_2})$
$\rightarrow\ \ If\ (Cons^{\mathtt{cons}}\ X^{\mathtt{z}}\ (Filter\ Nz\ XS))\ (Filter\ Nz\ XS)\ (Nz\ X^{\mathtt{s}\ v_0})$
$\rightarrow^*\ If\ (Cons^{\mathtt{cons}}\ X^{\mathtt{z}}\ (Filter\ Nz\ XS))\ (Filter\ Nz\ XS)\ (Nz\ (\mathtt{s}\ N^{v_0}))$
$\rightarrow\ \ If\ (Cons^{\mathtt{cons}}\ X^{\mathtt{z}}\ (Filter\ Nz\ XS))\ (Filter\ Nz\ XS)\ True^{\mathtt{true}}$
$\rightarrow^*\ Cons^{\mathtt{cons}}\ X^{\mathtt{z}}\ (Filter\ Nz\ XS)$
$\rightarrow\ \ \mathtt{cons}\ X^{\mathtt{z}}\ (Filter\ Nz\ XS)$
$\rightarrow^*\ \mathtt{cons}\ \mathtt{z}\ (Filter\ Nz\ XS)$

After labelling, we have $\mathsf{Failures} = \{(Filter, \{\mathtt{z}, \mathtt{s}\ v_0\})\}$. Observe that there are no labels on N in line 2, due to the fact that $\mathsf{labelTm}$ failed.

Feasibility. For a trace α in $\widetilde{\mathcal{P}_\mathcal{G}}$ to be feasible two properties are required. First, the non-determinism introduced by the abstraction should be well behaved and second, there should be a term in the input that is able to trigger the trace, i.e. when given as an argument to $Main$, the rest of the trace follows. The first of these conditions is the subject of the step case in $\mathsf{labelSeq}$, the second is the subject of the base case. Hence, if after performing $\mathsf{labelSeq}(\alpha)$ it is the case that $\mathsf{Failures} = \emptyset$, then we say α is *feasible*. The justification is the following lemma.

Lemma 8. *Let α be a feasible reduction sequence in $\widetilde{\mathcal{P}_\mathcal{G}}$ with error-witness q. Then there exists a term $t \in \mathcal{L}(\mathcal{G})$ and a finite reduction sequence $Main\ t \Rightarrow \cdots$ in $\mathcal{P}$ with error witness q.*

The witness to soundness, which appears in the proof of Theorem 2, will always be feasible. We say that any trace that is not feasible is *spurious*.

5.2 Refinement

When a reduction sequence in $\widetilde{\mathcal{P}_\mathcal{G}}$ is shown to be spurious, the problem can always be traced back to an occurrence of pattern-matching (notice that, by definition, the single parameter of the defining rule for $Main$ is always a pattern). Since the only loss of accuracy in the abstraction is in the way that data bound in pattern matches is handled during reduction, our remedy for infeasibility is to increase precision in the pattern matching rules of $\widetilde{\mathcal{P}_\mathcal{G}}$.

Our strategy is based on the observation that, due to the particular way in which the abstract wPMRS is constructed from the composite PMRS, the terminal symbol-labelled parts of each pattern are accurately preserved in the RHS of the defining rules of the abstraction. Based on the depth of pattern matches in the counterexample trace, we unfold patterns in the defining rules of $\mathcal{P}$ in a way that preserves the the set of possible reduction sequences.

Pattern-matching depth. To determine how much to unfold we define a measure $\mathsf{depth} : T(\Sigma, \mathcal{N}, \mathcal{V}) \rightarrow \mathbb{N}$, which quantifies the extent to which a term can be matched, as follows:

$$\mathsf{depth}(x) = 0$$
$$\mathsf{depth}(F\ t_1\ \cdots\ t_m) = 1$$
$$\mathsf{depth}(a\ t_1\ \cdots\ t_m) = 1 + \bigsqcup_{\mathbb{N}} \{\, \mathsf{depth}(t_i) \mid 1 \leq i \leq m \,\}$$

Given a set of non-terminals $\mathcal{N}$, a *depth profile* for $\mathcal{N}$ is a map $\mathcal{N} \rightarrow \mathbb{N}$. We assign a depth profile to a set of rules to quantify, for each non-terminal F, how accurately the defining rules for F model pattern-matching. Given a set of rules $\mathcal{R}$ defining non-terminals from $\mathcal{N}$, let the depth profile of $\mathcal{R}$, denoted $\mathsf{dp}(\mathcal{R})$, be the function:

$$\mathsf{dp}(\mathcal{R})(F) = \bigsqcup_{\mathbb{N}} \{\, \mathsf{depth}(p) \mid F\ x_1\ \cdots\ x_m\ p \longrightarrow t \in \mathcal{R} \,\}$$

Depth profiles can be naturally ordered pointwise, so that if d and d' are depth profiles over the same domain $\mathcal{N}$, then $d \leq d'$ iff $d(F) \leq d'(F)$ for all $F \in \mathcal{N}$.

Unfolding. To capture the result of unfolding we first introduce two auxiliary definitions. To aid readability, in each of them we will annotate fresh variables with their implied types by a superscript. The set of *atomic patterns* of type b, $\mathbb{A}_b$ is the set $\{\, a\ z_1^{b_1}\ \cdots\ z_{\mathsf{ar}(a)}^{b_{\mathsf{ar}(a)}} \mid a : b_1 \rightarrow \cdots \rightarrow b_{\mathsf{ar}(a)} \rightarrow b \in \Sigma \,\}$. For each $n \in \mathbb{N}$ we define the non-overlapping, exhaustive set of patterns of type b and depth n, $\mathsf{pats}_b(n)$:

$$\mathsf{pats}_b(0) = \{z^b\}$$
$$\mathsf{pats}_b(n+1) = \{\, p[q_1, \ldots, q_m / x_1^{b_1}, \cdots, x_m^{b_m}] \mid \varphi \,\}$$

where φ stands for the conjunction:

$$p \in \mathsf{pats}_b(n)\ \ \&\ \ \mathsf{FV}(p) = \{x_1^{b_1}, \ldots, x_m^{b_m}\}\ \ \&\ \ \forall i\, q_i \in \mathbb{A}_{b_i}$$

Hence, the depth 2 family of patterns of type $\mathbf{natlist}$ are given by (n, x and xs arbitrary variables):

$$\mathtt{nil},\quad \mathtt{cons}\ \mathtt{z}\ \mathtt{nil},\quad \mathtt{cons}\ \mathtt{z}\ (\mathtt{cons}\ x\ xs),$$
$$\mathtt{cons}\ (\mathtt{s}\ n)\ \mathtt{nil},\quad \mathtt{cons}\ (\mathtt{s}\ n)\ (\mathtt{cons}\ x\ xs)$$

To unfold the rules of a PMRS $\mathcal{P}$ according to a depth profile d, one constructs a new PMRS $\mathcal{P}'$ whose rule-set is enlarged so that, for a given non-terminal F of type $\tau_1 \rightarrow \cdots \rightarrow \tau_m \rightarrow b \rightarrow o$, there is a number of defining rules which is equal to the number of patterns of type b and depth $d(F)$. For each of these rules the corresponding right-hand side is constructed by using the existing $\mathcal{P}$ rules as a template.

Let $\mathcal{P} = \langle \Sigma, \mathcal{N}, \mathcal{R}, Main \rangle$ be a PMRS and let d be a depth profile with domain $\mathcal{N}$ such that $\mathsf{dp}(\mathcal{R}) \leq d$. The d-unfolding of $\mathcal{P}$ is the PMRS $\langle \Sigma, \mathcal{N}, \mathcal{R}', Main \rangle$, where $\mathcal{R}'$ is the set such that, for all substitutions σ, $F\ x_1\ \cdots\ x_m\ \sigma p \longrightarrow \sigma t \in \mathcal{R}'$ iff:

(i) $F\ x_1\ \cdots\ x_m\ p \longrightarrow t \in \mathcal{R}$
(ii) and p is of type b
(iii) and $q \in \mathsf{pats}_b(d(F))$
(iv) and $q = \sigma p$

Example 11. Let $\mathcal{P}$ be as in Example 2 and let d be the depth profile given by the following rule:

$$d(F) = \begin{cases} 2 & \text{when } F = Filter \\ \mathsf{dp}(\mathcal{R})(F) & \text{otherwise} \end{cases}$$

Then the d-unfolding of $\mathcal{P}$ is the PMRS $\mathcal{P}'$, whose rules are the same as $\mathcal{P}$ except that the two rules for $Filter$ have been replaced by the five rules in Figure 3.

Consider an abstraction $\widetilde{\mathcal{P}'_\mathcal{G}}$ of the PMRS $\mathcal{P}'$ in Example 11 (with $\mathcal{G}$ as given in Example 2). The only non-determinism that is introduced in constructing the abstraction is in replacing the pattern-matching variables in the right-hand sides of the defining rules by pattern-symbols. Due to the unfolding of the $Filter$ rules in $\mathcal{P}'$ (and hence in $\widetilde{\mathcal{P}'_\mathcal{G}}$), there is no longer a possibility to make the problematic reduction:

$Filter\ (Nz\ (\mathtt{cons}\ N\ ListN)$
$\rightarrow\ \ If\ (Cons\ X\ (Filter\ Nz\ XS))\ (Filter\ Nz\ XS)\ (Nz\ X)$

since the unfolded rules require more of the non-determinism (in the non-terminal symbols N and $ListN$) to be resolved earlier.

$$
\begin{aligned}
\mathit{Filter}\ p\ \mathtt{nil} &\longrightarrow \mathtt{nil}\\
\mathit{Filter}\ p\ (\mathtt{cons}\ \mathtt{z}\ \mathtt{nil}) &\longrightarrow \mathit{If}\ (\mathtt{cons}\ \mathtt{z}\ (\mathit{Filter}\ p\ \mathtt{nil}))\ (\mathit{Filter}\ p\ \mathtt{nil})\ (p\ \mathtt{z})\\
\mathit{Filter}\ p\ (\mathtt{cons}\ \mathtt{z}\ (\mathtt{cons}\ v_0\ v_1)) &\longrightarrow \mathit{If}\ (\mathtt{cons}\ \mathtt{z}\ (\mathit{Filter}\ p\ (\mathtt{cons}\ v_0\ v_1)))\ (\mathit{Filter}\ p\ (\mathtt{cons}\ v_0\ v_1))\ (p\ \mathtt{z})\\
\mathit{Filter}\ p\ (\mathtt{cons}\ (\mathtt{s}\ v_2)\ \mathtt{nil}) &\longrightarrow \mathit{If}\ (\mathtt{cons}\ (\mathtt{s}\ v_2)\ (\mathit{Filter}\ p\ \mathtt{nil}))\ (\mathit{Filter}\ p\ \mathtt{nil})\ (p\ (\mathtt{s}\ v_2))\\
\mathit{Filter}\ p\ (\mathtt{cons}\ (\mathtt{s}\ v_3)\ (\mathtt{cons}\ v_4\ v_5)) &\longrightarrow \mathit{If}\ (\mathtt{cons}\ (\mathtt{s}\ v_3)\ (\mathit{Filter}\ p\ (\mathtt{cons}\ v_4\ v_5)))\ (\mathit{Filter}\ p\ (\mathtt{cons}\ v_4\ v_5))\ (p\ (\mathtt{s}\ v_3))
\end{aligned}
$$

Figure 3. Depth-2 unfolding of the defining rules for Filter.

Refinement. Given a PMRS $\mathcal{P}$ and an infeasible error trace α in the abstraction of $\mathcal{P}$, we can obtain refined abstractions by unfolding the rules of $\mathcal{P}$ according to the depths of terms in the Failures set, then using the unfolded PMRS as the input to the next cycle of the abstraction-refinement loop.

Lemma 9. *Let $\mathcal{P} = \langle \Sigma, \mathcal{N}, \mathcal{R}, \mathit{Main}\rangle$ be a PMRS and $\widetilde{\mathcal{P}_\mathcal{G}}$ be the abstraction of $\mathcal{P}$ (starting from terms in $\mathcal{L}(\mathcal{G})$). Let α be a counterexample trace of $\widetilde{\mathcal{P}_\mathcal{G}}$ which is spurious with* Failures *set S. Let d be the depth profile with domain $\mathcal{N}$ defined by:*

$$d(F) = \mathsf{dp}(\mathcal{R})(F) + \bigsqcup_{\mathbb{N}} \{\, \mathsf{depth}(t) \mid (F,\, P) \in S,\, t \in P \,\}$$

and let $\mathcal{P}'$ be the d-unfolding of $\mathcal{P}$. Then α is not a reduction sequence in the abstraction $\widetilde{\mathcal{P}'_\mathcal{G}}$ of $\mathcal{P}'$.

Although it is clear that, given any spurious trace in some abstraction $\widetilde{\mathcal{P}_\mathcal{G}}$, it is possible to construct a refinement that eliminates it from any future abstraction $\widetilde{\mathcal{P}'_\mathcal{G}}$, the set of traces of $\widetilde{\mathcal{P}_\mathcal{G}}$ and the set of traces of $\widetilde{\mathcal{P}'_\mathcal{G}}$ are incomparable since, in general, there are new pattern-variables introduced in the refinement and hence new pattern-symbols into $\widetilde{\mathcal{P}'_\mathcal{G}}$. However, there is a very close relationship between the depth of a PMRS and the feasibility of reduction sequences in its abstraction.

Lemma 10. *Fix $n \in \mathbb{N}$. Then given any PMRS $\mathcal{P}$ and input grammar $\mathcal{G}$, there is a depth-profile d such that, if $\widetilde{\mathcal{P}'_\mathcal{G}}$ is the abstraction of the d-unfolding of $\mathcal{P}$, then all length-$m \leq n$ reduction sequences in $\widetilde{\mathcal{P}'_\mathcal{G}}$ are feasible.*

A consequence of this close relationship between depth and feasibility is that, under the assumption that the model-checker always reports the shortest counterexample trace, if the PMRS $\mathcal{P}$ (when run from a term in $\mathcal{I}$) does violate the property then eventually the abstraction-refinement cycle will produce a feasible counterexample trace demonstrating the fact.

Theorem 4 (Semi-completeness)**.** *Let $(\mathcal{P},\, \mathcal{I},\, \mathcal{A})$ be a no-instance of the verification problem. Then the algorithm terminates with a feasible counterexample trace.*

6. Related work

We compare and contrast our work with a number of topics in the literature broadly related to flow analysis and verification of functional programs.

Higher-order multi-parameter tree transducer. As discussed in the Introduction, Kobayashi [6] introduced a type-based verification method for temporal properties of higher-order functional programs generated from finite base types. In a follow-up paper [9], Kobayashi et al. introduced a kind of tree transducer, called HMTT, that uses pattern-matching, taking trees as input and returning an output tree. They studied the problem of whether the tree generated by a given HMTT meets the output specification, assuming that the input trees meet the input specification, where both input and output specifications are regular tree languages. A sound but incomplete algorithm has been proposed for the HMTT verification problem by reduction to a model checking problem for recursion schemes with finite data domain (which can then be solved by a variation of Kobayashi's type-based algorithm). Though our algorithm in the present paper solves a similar kind of verification problem, it is not straightforward to compare it with the HMTT work [9]. It would appear that PMRS is a more general (and natural) formalism than HMTT. What is clear is that our approach to the over-approximation is very different: we use binding analysis to obtain a wPMRS which generates an over-approximation of the reachable term-trees, whereas Kobayashi et al. use automaton states to approximate input trees.

Approximating collecting semantics and flow analysis. In a seminal paper [5], Jones and Andersen studied the (data) flow analysis of functional programs by safely approximating the behaviour of a certain class of untyped, first-order, term rewriting systems with pattern matching. Their algorithm takes a regular set $\mathcal{I}$ of input terms, a program $\mathcal{P}$ and returns a regular tree grammar which is a "safe" description of the set of all reachable (constructor) terms of the computation of $\mathcal{P}$ with inputs form $\mathcal{I}$. Precisely, the algorithm computes a safe approximation of the *collecting semantics* of $\mathcal{P}$ relative to $\mathcal{I}$, which assigns to each rewrite rule a set of pairs (θ, g_θ) such that θ is a substitution (*realisable* in the course of such a computation) of actual parameters to the formal parameters of the rule, and g_θ is a term reachable from the RHS of the rule with the substitution θ. The collecting semantics is undecidable in general. Jones and Andersen was able to obtain, for each rewrite rule, a regular over-approximation of the set of realisable bindings $\{\, x \mapsto \theta\, x \mid \text{ realisable } \theta \,\}$ for each formal parameter x of the rule, and the set of reachable terms $\{\, g_\theta \mid \text{ realisable } \theta \,\}$, by decoupling the pair (θ, g_θ).

There are two directions in which Jones and Andersen's algorithm may be refined. Consider the setting of simply-typed functional programs with pattern-matching algebraic data types. Recent advances in the model checking of higher-order recursion schemes (notably the decidability of MSO theories of trees generated by higher-order recursion schemes [14]) indicate that the bindings of *non* pattern-matching variables, whether higher-order or not, can be precisely analysed algorithmically (though with extremely high asymptotic complexity). Jones and Andersen's algorithm builds a regular approximation of the binding set of every variable. A natural question is whether one can improve it by approximating *only* the bindings of pattern-matching variables, while analysing other variables (including all higher-order variables) precisely using the method in [14]. The work presented here offers a positive answer to the question. Another direction worth investigating is to seek to preserve, for each rewrite rule, as much of the connection between realisable substitutions θ and reachable terms g_θ as one can get away with. In an recent dissertation [10], Kochems has presented just such an algorithm using a kind of linear indexed tree grammars

(which are equivalent to context-free tree grammars) whereby the indices are the realisable substitutions.

To compare our algorithm with Jones and Andersen's, it is instructive to apply their algorithm to our Example 8. Their framework can be extended to simply-typed and higher-order programs. It is an old idea in functional programming that an higher-order expression, such as an "incompletely applied" function $(\cdots(f\,e_1)\cdots)e_m$ where the type of f has arity greater than m, may be viewed as a closure. (Indeed, closures are a standard implementation technique.) From this viewpoint, a higher-order non-terminal is regarded, not as a defined operator, but as a constructor, and closures are formed using a *binary closure-forming operator* @. Thus, the second clause of $Map2$ is written in their system as

$$@\ (@\ (@\ Map2\ \varphi)\ \psi)\ (\texttt{cons}\ x\ xs) \longrightarrow \texttt{cons}\ (@\ \varphi\ x)\ (@\ (@\ (@\ Map2\ \psi)\ \varphi)\ xs)$$

Observe that in this setting, $Map2$ is a constructor (i.e. terminal) symbol, and the expression $(@\ (@\ Map2\ \varphi)\ \psi)$ a pattern. Call the *binding set* of a variable the set of terms that may be bound to it at some point in the course of a computation. The approximating grammar produced by Jones and Andersen's algorithm is always regular (equivalently an order-0 recursion scheme). This is achieved by over-approximating the binding set of *every* variable (including higher-order ones, such as φ). The resultant grammar generates all finite lists of 0's and 1's, which is less precise than our algorithm.

Control flow analysis. Established in the 80's by Jones [4], Shivers [17] and others, *Control Flow Analysis (CFA)* of functional programs has remained an active research topic ever since (see e.g. Midtgaard's survey [12] and the book by Nielson et al. [13]). The aim of CFA is to approximate the flow of control within a program phrase in the course of a computation.

In a functional computation, control flow is determined by a sequence of function calls (possibly unknown at compile time); thus CFA amounts to approximating the values that may be substituted for bound variables during the computation. Since these values are (denoted by) pieces of syntax, CFA reduces to an algorithm that assigns *closures* (subterms of the examined term paired with substitutions for free variables) to bound variables. Reachability analysis and CFA are clearly related: for example, the former can aid the latter because unreachable parts of the term can be safely excluded from the range of closure assignment. There are however important differences: on one hand, CFA algorithms are *approximation algorithms* designed to address a more general problem; on the other, because CFA considers terms in isolation of its possible (program) contexts, the corresponding notion of reachability essentially amounts to reachability in the reduction graph.

Functional reachability. Based on the fully abstract game semantics, *traversals* [1, 14] are a (particularly accurate) model of the flow of control within a term; they can therefore be viewed as a CFA method. Using traversals, a new notion of reachability of higher-order functional computation (in the setting of PCF) is studied in [16], called *Contextual Reachability*: given a PCF term M of type A and a subterm N^α with occurrence α, is there a program context $C[-]$ such that $C[M]$ is a closed term of ground type and the evaluation of $C[M]$ causes control to flow to N^α?

7. Conclusion

Recursion schemes with pattern matching (PMRS) are an accurate and natural model of computation for functional programs have pattern-matching algebraic data types. We have given an algorithm that, given a PMRS $\mathcal{P}$ and a regular set $\mathcal{I}$ of input terms, constructs a *recursion scheme with weak pattern-matching* (wPMRS) that (i) over-approximates the set of terms reachable from under rewriting from $\mathcal{P}$ (ii) has a decidable model checking problem (relative to trivial automata). Finally, because of the precise analysis at higher-orders, we show that there is a simple notion of automatic abstraction-refinement, which gives rise to a semi-completeness property.

For future work, we plan to build an implementation of the verification algorithm for a real functional programming language. We shall be especially interested in investigating the scalability of our approach.

Acknowledgements. We would like to thank the anonymous reviewers for many useful comments.

References

[1] W. Blum and C.-H. L. Ong. Path-correspondence theorems and their applications. Preprint, 2009.

[2] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *CAV '00: Proceedings of the 12th International Conference on Computer Aided Verification*, pages 154–169, London, UK, 2000. Springer-Verlag.

[3] A. Igarashi and N. Kobayashi. Resource usage analysis. *ACM Trans. Program. Lang. Syst.*, 27(2):264–313, 2005.

[4] N. D. Jones. Flow analysis of lambda expressions (preliminary version). In *Proceedings of the 8th Colloquium on Automata, Languages and Programming*, pages 114–128. Springer-Verlag, 1981. ISBN 3-540-10843-2.

[5] N. D. Jones and N. Andersen. Flow analysis of lazy higher-order functional programs. *Theoretical Computer Science*, 375:120–136, 2007.

[6] N. Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *Proceedings of POPL 2009*, pages 416–428. ACM Press, 2009.

[7] N. Kobayashi. Model-checking higher-order functions. In *PPDP*, pages 25–36, 2009.

[8] N. Kobayashi and C.-H. L. Ong. A type theory equivalent to the modal mu-calculus model checking ofhigher-order recursion schemes. In *Proceedings of LICS 2009*. IEEE Computer Society, 2009.

[9] N. Kobayashi, N. Tabuchi, and H. Unno. Higher-order multi-parameter tree transducers and recursion schemes for program verification. In *POPL*, pages 495–508, 2010.

[10] J. Kochems. Approximating reachable terms of functional programs. University of Oxford MMathsCompSc thesis, 2010.

[11] R. P. Kurshan. *Computer Aided Verification of Coordinating Processes*. Princeton University Press, 1994.

[12] J. Midtgaard. Control-flow analysis of functional programs. Technical Report BRICS RS-07-18, DAIMI, Department of Computer Science, University of Aarhus, Aarhus, Denmark, Dec 2007. URL `http://www.brics.dk/RS/07/18/BRICS-RS-07-18.pdf`.

[13] F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer-Verlag New York, 1999.

[14] C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *Proceedings 21st Annual IEEE Symposium on Logic in Computer Science, Seattle*, pages 81–90. Computer Society Press, 2006. Long version (55 pp.) downloadable at `users.comlab.ox.ac.uk/luke.ong/`.

[15] C.-H. L. Ong and S. J. Ramsay. Verifying higher-order functional programs with pattern-matching algebraic data types. Long version, available from: `https://mjolnir.comlab.ox.ac.uk/papers/pmrs.pdf`.

[16] C.-H. L. Ong and N. Tzevelekos. Functional reachability. In *LICS*, pages 286–295, 2009.

[17] O. Shivers. *Control-flow analysis of higher-order languages*. PhD thesis, Carnegie-Mellon University, 1991.