Improved Functional Flow and Reachability Analyses Using Indexed Linear Tree Grammars

Jonathan Kochems and Luke Ong

Oxford University Computing Laboratory

Abstract

The collecting semantics of a program defines the strongest static property of interest. We study the analysis of the collecting semantics of higher-order functional programs, cast as left-linear term rewriting systems. The analysis generalises functional flow analysis and the reachability problem for term rewriting systems, which are both undecidable. We present an algorithm that uses *indexed linear tree grammars* (ILTGs) both to describe the input set and compute the set that approximates the collecting semantics. ILTGs are equi-expressive with pushdown tree automata, and so, strictly more expressive than regular tree grammars. Our result can be seen as a refinement of Jones and Andersen's procedure, which uses regular tree grammars. The main technical innovation of our algorithm is the use of indices to capture (sets of) substitutions, thus enabling a more precise binding analysis than afforded by regular grammars. We give a simple proof of termination and soundness, and demonstrate that our method is more accurate than other approaches to functional flow and reachability analyses in the literature.

Keywords and phrases Flow analysis, reachability, collecting semantics, higher-order program, term rewriting, indexed linear tree grammar

Digital Object Identifier 10.4230/LIPIcs.RTA.2011.187

Category Regular Research Paper

1 Introduction

In program analysis, the *collecting semantics* of a program maps a given program point to the collection of all states attainable by a run of the program when control reaches that point. Thus the collecting semantics defines the strongest static property of interest [3]. A good method of analysing the collecting semantics of programs is the basis of a useful *generic* tool; it can be employed, *a fortiori*, to analyse such practically important computational properties as reachability and control flow.

A higher-order functional program with pattern-matching algebraic data types may be viewed[1] as a (left-linear) term rewriting system, namely, a set $\mathcal{P} = \{ l_i \to r_i \mid 1 \le i \le p \}$ of rewrite rules where each l_i and r_i are elements of the term algebra of a given signature, generated from a set of variables. The one-step rewrite relation, $\to_{\mathcal{P}}$, is standard: if a ground term t has a subterm u that matches the pattern l_i (i.e. $t = C[u]$ for some context $C[\text{-}]$, and $\sigma\, l_i = u$ for some ground substitution σ) then $t = C[\sigma\, l_i]$ rewrites in one step to $C[\sigma\, r_i]$, written $C[\sigma\, l_i] \to_{\mathcal{P}} C[\sigma\, r_i]$. What then is the collecting semantics of $\mathcal{P}$? Following Jones and Andersen [10], we take the program points of $\mathcal{P}$ to be the rewrite rules, and the

[1] The idea goes back to Reynolds' *defunctionalization* [20]. A higher-order lambda-term can be systematically "lambda-lifted" to an equivalent term rewriting system that has explicit binary application operators, systematically replacing closed higher-order lambda-terms by named combinators; thus every function is treated as "curried". See, for example, Jones and Andersen's account of the translation [10].

22nd International Conference on Rewriting Techniques and Applications.
Editor: M. Schmidt-Schauß; pp. 187–202
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

states of $\mathcal{P}$ to be the ground substitutions. The *collecting semantics* of $\mathcal{P}$ over a set I of input terms is then the tuple $(Z_0, Z_1, \ldots, Z_p)$, where

$$Z_0 \coloneqq \{\, (Id, t) \mid \exists s \in I \,.\, s \to_{\mathcal{P}}^* t \,\}$$
$$Z_i \coloneqq \{\, (\sigma, t) \mid \exists s \in I \,.\, s \to_{\mathcal{P}}^* C[\sigma\, l_i] \;\wedge\; \sigma\, r_i \to_{\mathcal{P}}^* t \,\} \quad \text{for } 1 \le i \le p$$

In words, Z_i consists of all pairs (σ, t) where σ is a substitution that matches the LHS of rule i in a $\mathcal{P}$-computation which is reachable from I (we call σ an *I-reachable substitution*), and t is a *result term* which is reachable from the RHS of rule i when instantiated by σ. (In a functional computation, control flow is determined by a sequence of function calls, possibly unknown at compile time. Thus, the flow analysis of $\mathcal{P}$ amounts to approximating the values that may be substituted for the program variables of each rewrite rule during a run of $\mathcal{P}$. It follows that one can use collecting semantics to flow-analyse functional programs.) Since a precise characterisation of the collecting semantics is uncomputable, our goal is to build over-approximations of the Z_is.

The key to our construction is a binding analysis that uses indices to explicitly model (sets of) I-reachable substitutions in the setting of *indexed linear tree grammars* (ILTGs). The rules of an ILTG rewrite term-trees in which leaf-nodes may be labelled by a non-terminal annotated with a sequence of indices; indices propagate from the root to the leaves of a term-tree. ILTGs are strictly more expressive than regular tree grammars; in fact, they are equi-expressive with pushdown tree automata. We give an algorithm which takes three input arguments (namely, a program $\mathcal{P}$ as before, a set of input terms defined by an ILTG $\mathcal{G}_0$, and an accuracy parameter $n \ge 0$) and constructs an ILTG $\mathcal{G}^n$ that approximates the collecting semantics of $\mathcal{P}$ on input given by $\mathcal{G}_0$. Precisely, the ILTG $\mathcal{G}^n$, which is equipped with distinguished non-terminals

- R_i (denoting the "results of the $\mathcal{P}$-rule, $l_i \to r_i$"), for each $1 \le i \le p$, and
- X, for each program *variable* X,

over-approximates Z_i (for each $0 \le i \le p$) in the following sense:

> *local safety*: for every $(\sigma, t) \in Z_i$, there is (constructively) an index sequence δ_σ such that $R_i\, \delta_\sigma \to_{\mathcal{G}^n}^* t$ and $X\, \delta_\sigma \to_{\mathcal{G}^n}^* \sigma X$ for each variable X that occurs in r_i

where σX denotes the term obtained by applying the substitution σ to the *variable* X, and $\to_{\mathcal{G}^n}$ is the one-step rewrite relation of $\mathcal{G}^n$. The superscript n of $\mathcal{G}^n$ controls the accuracy of approximation: if $n < n'$ then the ILTG $\mathcal{G}^{n'}$ offers at least as accurate an approximation as $\mathcal{G}^n$. Happily, the ILTG $\mathcal{G}^n$ is not prohibitively large: its size is polynomial in the number of rules in $\mathcal{P}$ and $\mathcal{G}_0$, and exponential in n and the number[2] of program variables in $\mathcal{P}$.

As far as we know, our algorithm gives the first completion procedure for indexed linear tree grammars (equivalently, pushdown tree automata). It extends Jones and Andersen's safe approximation of collecting semantics by admitting a strictly larger class of input sets. Even when restricted to regular input sets, for each $n \ge 0$, our algorithm builds an ILTG $\mathcal{G}^n$ which is at least as accurate as the result of Jones and Andersen's method.

Since the 2-projection of Z_0 is the set $\mathit{Reach}_{\mathcal{P}}(I)$ of terms that are reachable from the input set I under rewriting by $\mathcal{P}$, our analysis of collecting semantics may also be viewed as a contribution to the reachability problem for left-linear term rewriting systems (TRS) (i.e. given an input set I and a left-linear TRS $\mathcal{R}$, construct a set that over-approximates $\mathit{Reach}_{\mathcal{R}}(I)$) [9, 5, 2]. To the best of our knowledge, none of the over-approximation results in the literature can admit an arbitrary pushdown tree language as the input set.

---

[2] This can be improved to $\max_{1 \le i \le p} \#\mathit{vars}(l_i)$ where $\#\mathit{vars}(l_i)$ is the number of variables in l_i.

Outline In Section 2, after fixing notations we introduce indexed linear tree grammars and the notion of minimally reachable match. The ILTG completion algorithm for over-approximating the collecting semantics of a program on an input set is presented in Section 3. The termination proof and soundness proof are given in Sections 4 and 5 respectively. In the concluding section, we evaluate our result and set out a number of further directions. Note, a long version of the paper is available [13] containing the proofs that are omitted from the main text together with two complete worked examples which illustrate the workings of our algorithm.

The work reported here is based on preliminary results first presented in the first author's MSc dissertation [12].

2 Preliminaries

Term Rewriting Systems and Programs Let Σ be a ranked alphabet equipped with a function ar giving the arity of each symbol in Σ. We write $\Sigma_n := ar^{-1}(n)$ and use letters f, g, h, a and b to denote members of Σ. The *free algebra* over an arbitrary set $\mathcal{X}$, written $\mathcal{T}_\Sigma(\mathcal{X})$, is the smallest set such that $\mathcal{X} \subseteq \mathcal{T}_\Sigma(\mathcal{X})$, and if $f \in \Sigma_n$ and $t_1, \ldots, t_n \in \mathcal{T}_\Sigma(\mathcal{X})$ then $f(t_1, \ldots, t_n) \in \mathcal{T}_\Sigma(\mathcal{X})$. Elements of $\mathcal{T}_\Sigma(\mathcal{X})$ are called *terms*, denoted by letters s and t; and we write $\mathcal{T}_\Sigma := \mathcal{T}_\Sigma(\varnothing)$ for the set of *ground terms*.

Let $\mathcal{V} = \{ X, Y, Z, \cdots \}$ be a set of variables, and $\Sigma = \Delta \uplus \Gamma$ be a ranked alphabet that is partitioned into disjoint sets Δ and Γ. Symbols in Δ and Γ are called *defined operators* and *constructors* respectively. A *call* is a term of the form $f(t_1, \ldots, t_n)$ with $f \in \Delta$; a *pattern* is a call in which every variable occurs at most once. A *term rewriting system (TRS) over* Σ is a finite set of rewrite rules $\mathcal{P} = \{l_i \to r_i \mid l_i, r_i \in \mathcal{T}_\Sigma(\mathcal{V}); 1 \le i \le p\}$ such that for every i, l_i is a call, and every variable that occurs in r_i also occurs in l_i; further $\mathcal{P}$ is *left-linear* just if for each i, l_i is a pattern. The (one-step) *rewrite relation* $\to_\mathcal{P} \subseteq (\mathcal{T}_\Sigma(\mathcal{X}))^2$ of a TRS $\mathcal{P}$ is defined as

$$C[\sigma\, l] \to_\mathcal{P} C[\sigma\, r]$$

with $C[\text{-}]$ ranging over one-hole contexts, $\sigma : \mathcal{V} \to \mathcal{T}_\Sigma$ ranging over substitutions, and $l \to r$ ranging over rules in $\mathcal{P}$. The n-step rewrite relation, $\to^n_\mathcal{P}$, and the reflexive, transitive closure, $\to^*_\mathcal{P}$, of $\to_\mathcal{P}$ are defined in the usual way.

Henceforth we fix a ranked alphabet $\Sigma = \Delta \uplus \Gamma$. By a *program* we mean a pair $(\Sigma, \mathcal{P})$ where $\mathcal{P}$ is a left-linear TRS over Σ. (We can think of the defined operators and constructors respectively as the non-terminals and terminals of the program.) An *input* of $\mathcal{P}$ is a set $I \subseteq \mathcal{T}_\Sigma$ of ground terms. We are interested in analysing (*inter alia*) the set $\mathit{Reach}_\mathcal{P}(I) := \{t \mid s \in I, s \to^*_\mathcal{P} t\}$ of *$\mathcal{P}$-reachable terms from I*.

▸ **Example 2.1** (Running Example)**.** Consider the program $\mathcal{P}$ with $\Delta_1 = \{\mathsf{counter}, \mathsf{genh}, \mathsf{genk}\}$, $\Gamma_0 = \{\mathsf{0}, \mathsf{a}, \mathsf{b}\}$, $\Gamma_1 = \{\mathsf{S}, \mathsf{h}, \mathsf{k}\}$, $\Gamma_2 = \{\mathsf{f}\}$, and rules as follows:

1. $\mathsf{counter}(\mathsf{x}) \to \mathsf{counter}(\mathsf{S}(\mathsf{x}))$
2. $\mathsf{counter}(\mathsf{x}) \to \mathsf{f}(\mathsf{genh}(\mathsf{x}), \mathsf{genk}(\mathsf{x}))$
3. $\mathsf{genh}(\mathsf{0}) \to \mathsf{a}$
4. $\mathsf{genh}(\mathsf{S}(\mathsf{x})) \to \mathsf{h}(\mathsf{genh}(\mathsf{x}))$
5. $\mathsf{genk}(\mathsf{0}) \to \mathsf{b}$
6. $\mathsf{genk}(\mathsf{S}(\mathsf{x})) \to \mathsf{k}(\mathsf{genk}(\mathsf{x}))$

Let $I = \{\, \mathsf{counter}(\mathsf{0}) \,\}$. The set of reachable constructor-terms from I, namely, $\mathit{Reach}_\mathcal{P}(I) \cap \mathcal{T}_\Gamma$, is $\{\, \mathsf{f}(\mathsf{h}^n(\mathsf{a}), \mathsf{k}^n(\mathsf{b})) \mid n \ge 0 \,\}$.[3]

[3] There is a simpler program that gives the same set of reachable constructor terms, namely,

Indexed Linear Tree Grammar (ILTG) Jones and Andersen's algorithm constructs a regular tree grammar to over-approximate the collecting semantics of a given program $\mathcal{P}$ on input I. Our refinement works in a similar fashion but builds an indexed linear tree grammar instead. Let us introduce ILTG with an example.

▸ **Example 2.2.** Consider the grammar with non-terminal alphabet $\mathcal{N} = \{\mathsf{S},\mathsf{S'},\mathsf{A},\mathsf{B}\}$, terminal alphabet $\Sigma = \{\mathsf{f},\mathsf{h},\mathsf{k},\mathsf{a},\mathsf{b}\}$ and index set $\mathcal{F} = \{\alpha,\beta\}$. The rewrite rules are as follows.

1. $\mathsf{S} \to \mathsf{S'}\,\beta$
2. $\mathsf{S'} \to \mathsf{S'}\,\alpha$
3. $\mathsf{S'} \to \mathsf{f}(\mathsf{A},\mathsf{B})$
4. $\mathsf{A}\alpha \to \mathsf{h}(\mathsf{A})$
5. $\mathsf{A}\beta \to \mathsf{a}$
6. $\mathsf{B}\alpha\,\alpha \to \mathsf{k}(\mathsf{k}(\mathsf{B}))$
7. $\mathsf{B}\alpha\,\beta \to \mathsf{k}(\mathsf{b})$
8. $\mathsf{B}\beta \to \mathsf{b}$

This ILTG rewrites terms in $\mathcal{T}_\Sigma(\mathcal{N}\mathcal{F}^*)$. For example, the rule $\mathsf{B}\alpha\,\alpha \to \mathsf{k}(\mathsf{k}(\mathsf{B}))$ allows us to replace the term $\mathsf{B}\alpha\,\alpha\,\nu_1 \ldots \nu_n$ by the term $\mathsf{k}(\mathsf{k}(\mathsf{B}\nu_1 \ldots \nu_n))$ where $\nu_1 \ldots \nu_n \in \mathcal{F}^*$ —note the propagation of $\nu_1 \ldots \nu_n$ from the root of the term-tree $\mathsf{k}(\mathsf{k}(\mathsf{B}))$ to its leaf. Similarly Rule 4 allows the term $\mathsf{A}\alpha\,\beta$ to rewrite to $\mathsf{h}(\mathsf{A}\beta)$. A possible rewrite using the above rules is

$$\mathsf{S'}\,\alpha\,\beta \to \mathsf{S'}\,\alpha\,\alpha\,\beta \to \mathsf{f}(\mathsf{A}\alpha\,\alpha\,\beta\,,\mathsf{B}\alpha\,\alpha\,\beta\,) \to \mathsf{f}(\mathsf{A}\alpha\,\alpha\,\beta\,,\mathsf{k}(\mathsf{k}(\mathsf{B}\beta))).$$

In this ILTG, the set of reachable ground terms (in $\mathcal{T}_\Sigma$) from input $\{S\}$ of the ILTG is $\{\mathsf{f}(\mathsf{h}^n(\mathsf{a}),\mathsf{k}^n(\mathsf{b})) \mid n \geq 0\}$ which is the same as that of the TRS in Example 2.1. Note that the set is not regular; it follows that ILTGs are strictly more expressive than regular tree grammars. Note also the similarities between the computation of this ILTG and that of the TRS in Example 2.1.

We now give a formal definition of indexed linear tree grammars.

▸ **Definition 2.3** (ILTG)**.** An *indexed linear tree grammar (ILTG)* is a 5-tuple $(\Sigma,\mathcal{N},\mathcal{F},S,\mathcal{G})$ (of finite objects) where

- Σ is a ranked alphabet of terminal symbols (ranged over by f,g,a,b, etc.)
- $\mathcal{N}$ is an alphabet of nullary non-terminal symbols (ranged over by A,B,C, etc.), and $S \in \mathcal{N}$ is a distinguished *start* symbol
- $\mathcal{F}$ is a set of index symbols (ranged over by α,β, etc.)
- $\mathcal{G}$ a set of rewrite rules of the form $A\,\gamma \to_\mathcal{G} t$, where $A \in \mathcal{N}$, $\gamma \in \mathcal{F}^*$ and $t \in \mathcal{T}_\Sigma(\mathcal{N}\mathcal{F}^*)$.

By abuse of notation, we shall refer to the ILTG as $\mathcal{G}$. The (one-step) *rewrite relation* of $\mathcal{G}$, $\to_\mathcal{G} \subseteq (\mathcal{T}_\Sigma(\mathcal{N}\mathcal{F}^*))^2$, is defined as

$$C[A\,\gamma\,\delta] \to_\mathcal{G} C[\mathrm{dist}_\delta(t)]$$

with $C[\text{-}]$ ranging over (one-holed) contexts, δ over $\mathcal{F}^*$ and $A\,\gamma \to_\mathcal{G} t$ over rules in $\mathcal{G}$, and $\mathrm{dist}_\delta(t)$ is defined as $\mathrm{dist}_\delta(A\,\gamma) := A\,\gamma\,\delta$ and $\mathrm{dist}_\delta(f(t_1,\ldots,t_n)) := f(\mathrm{dist}_\delta(t_1),\ldots,\mathrm{dist}_\delta(t_n))$, where $n \geq 0$, and f, A and γ range over $\Sigma,\mathcal{N}$ and $\mathcal{F}^*$ respectively. For example,

$$\mathrm{dist}_{\alpha_1\alpha_2}(f(A\beta_1, g(B\beta_2\beta_3, a))) = f(A\beta_1\alpha_1\alpha_2, g(B\beta_2\beta_3\alpha_1\alpha_2, a)).$$

We denote the reflexive, transitive closure and the n-step rewrite relation of $\to_\mathcal{G}$ by $\to_\mathcal{G}^*$ and $\to_\mathcal{G}^n$ respectively.

$\mathsf{gen}(\mathsf{x},\mathsf{y}) \to \mathsf{gen}(\mathsf{h}(\mathsf{x}),\mathsf{k}(\mathsf{y})) \mid \mathsf{f}(\mathsf{x},\mathsf{y})$ with input $I = \{\mathsf{gen}(\mathsf{a},\ \mathsf{b})\}$.

Take an ILTG $(\Sigma, \mathcal{N}, \mathcal{F}, S, \mathcal{G})$, and let $I \subseteq \mathcal{T}_\Sigma(\mathcal{N}\mathcal{F}^*)$. We define $Reach_\mathcal{G}(I) := \{t \mid s \in I, s \to_\mathcal{G}^* t\}$; for singleton sets we omit set braces e.g. $Reach_\mathcal{G}(t)$ means $Reach_\mathcal{G}(\{t\})$; and if S is $\mathcal{G}$'s start symbol we write $Reach_\mathcal{G}$ for $Reach_\mathcal{G}(S)$. If $Reach_\mathcal{G}$ is well-defined we write $Reach_\mathcal{P}(\mathcal{G}) := Reach_\mathcal{P}(Reach_\mathcal{G})$ and set $Reach^o_\mathcal{G}(I) := Reach_\mathcal{G}(I) \cap \mathcal{T}_\Sigma$. Note that elements of $\mathcal{T}_\Sigma(\mathcal{N}\mathcal{F}^*)$ are term-trees: they can be viewed as (finite) ranked trees, whose internal nodes are labelled by symbols in Σ (of non-zero arities), and whose leaves are labelled by symbols in $\mathcal{N}\mathcal{F}^* \cup \Sigma_0$. Thus we define the *language of (finite) Σ-labelled trees generated by $\mathcal{G}$* to be $Reach^o_\mathcal{G}$.

▸ **Proposition 2.4.** *ILTGs are equi-expressive with pushdown tree automata [8] as generators of Σ-labelled tree languages. Thus ILTGs generate precisely the level-1 trees*[4] *of the hierarchy of (collapsible) pushdown trees [18].*

The idea is that the non-terminals and indices of an ILTG correspond respectively to the states and stack symbols of a pushdown tree automaton. See the full paper for a proof.

▸ Remark 2.5. (i) ILTGs are similar to Aho's indexed grammars [1] but there are important differences. First Aho's grammars are generators of word languages which are equi-expressive with second-order pushdown word automata; they define level 2 of the Maslov Hierarchy [16, 18]. Secondly, the linearity constraint on ILTG (each leaf of the RHS of a rule has at most one occurrence of a non-terminal) has the effect of reining in the power of indices, so that the branch language of a tree generated by an ILTG is context-free. (ii) Engelfriet and Vogler [4] introduced regular tree grammars with pushdown store which are equi-expressive with context-free tree languages [8], and with ILTGs. (iii) ILTGs define a class of trees with rich algorithmic properties, which make them highly suitable for verification (for example, their emptiness problem is in EXPTIME). In fact, all trees in the hierarchy of (collapsible) pushdown trees have decidable MSO theories [17].

ILTGs are a concise formalism; when rewriting, indices propagate from the root of a term-tree to its leaves, just like substitutions. We think that ILTGs are an attractive vehicle for the presentation of our algorithm.

Minimally Reachable Match Minimally reachable match is a concept due to Jones and Andersen [10]; it formalises the idea of rewriting a term by as many steps as necessary—but no more—in order to achieve a match against a pattern. Here we generalise it to the setting of indexed grammar, and consider the substitution that witnesses a minimally reachable match of a *uniformly indexed* term against a pattern. A term $t \in \mathcal{T}_\Sigma(\mathcal{N}\mathcal{F}^*)$ is said to be *uniformly indexed* (or simply *uniform*) just if $t = \mathrm{dist}_\delta(s)$ for some $\delta \in \mathcal{F}^*$ and $s \in \mathcal{T}_\Sigma(\mathcal{N})$ i.e. every non-terminal in t is annotated with the same index sequence.

▸ **Definition 2.6** (Minimally Reachable Match)**.** Let $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{F}, S, \mathcal{G})$ be an ILTG, $s \in \mathcal{T}_\Sigma(\mathcal{V})$ be a pattern, and $\mathrm{dist}_{\alpha_1 \ldots \alpha_n} t$ be a uniform term (thus $t \in \mathcal{T}_\Sigma(\mathcal{N})$). We say that a substitution σ is a *minimally reachable match of* $\mathrm{dist}_{\alpha_1 \ldots \alpha_n}(t)$ *against* s just if there exists $m \geq 0$ such that

(i) $\mathrm{dist}_{\alpha_1 \ldots \alpha_n}(t) \to_\mathcal{G}^m \sigma s$, and
(ii) $\neg(\exists \sigma'. \mathrm{dist}_{\alpha_1 \ldots \alpha_n}(t) \to_\mathcal{G}^{m-1} \sigma' s \to \sigma s)$, and
(iii) $\neg(\exists \sigma'. \exists k < n. (\mathrm{dist}_{\alpha_1 \ldots \alpha_k}(t) \to_\mathcal{G}^m \sigma' s$ and $\sigma = \mathrm{dist}_{\alpha_{k+1} \ldots \alpha_n}(\sigma')))$.

[4] See, for example, the survey [18] for an introduction to the hierarchies of finite and infinite ranked trees. Note that ILTGs are generators of (languages of) both finite and infinite trees.

I.e. m is the minimal number of reduction steps from $\mathrm{dist}_{\alpha_1 \ldots \alpha_n} t$, and $\alpha_1 \ldots \alpha_n$ is a corresponding minimal index sequence, that are required to achieve a match against the pattern s. We shall sometimes refer to σ as a *minimally reachable substitution*.

Condition (iii) means that every index in $\alpha_1 \ldots \alpha_n$ must be "consumed" in the construction of σ. For example, take the rules in Example 2.2 and the pattern $s = \mathsf{f(g(X),h(h(Y)))}$ then $\mathsf{f(A}\alpha\,\alpha\mathsf{,B}\alpha\,\alpha\mathsf{)} \to^* \mathsf{f(g(A}\alpha\mathsf{),h(h(B)))} = \sigma s$ is a minimal derivation, where $\sigma = \{\mathsf{X} \mapsto \mathsf{A}\alpha, \mathsf{Y} \mapsto \mathsf{B}\}$. However the derivation $\mathsf{f(A}\alpha\,\alpha\,\beta\mathsf{,B}\alpha\,\alpha\,\beta\mathsf{)} \to^* \mathsf{f(g(A}\alpha\,\beta\mathsf{),h(h(B}\beta\mathsf{)))} = \mathrm{dist}_\beta(\sigma s)$ is not minimal, because β is superfluous. Note that in the absence of indices (i.e. in a regular tree grammar), n is necessarily 0, and so, the notion of minimally reachable match here coincides with that of Jones and Andersen's.

We say that an ILTG is *uniform* if the RHS of every rule is uniform. In a uniform ILTG, a minimally reachable match of a uniform term t against a pattern p has the nice property that the substitution in question will only replace a variable with a subterm of t (which is uniform) or an indexed subterm of the RHS of a rule (which is also uniform). In the following we write $r' \leq r$ to mean that r' is a subterm of r.

▸ **Proposition 2.7.** *Let $\mathcal{G}$ be an ILTG, $t \in \mathcal{T}_\Sigma(\mathcal{N})$ and $p \in \mathcal{T}_\Sigma(\mathcal{V})$. If σ is the substitution of a minimally reachable match of $\mathrm{dist}_\gamma(t)$ against p then for all X in $Vars(p)$, $\sigma X \leq \mathrm{dist}_\gamma(t)$, or $\sigma X = \mathrm{dist}_{\gamma'}(g)$ for some index sequence γ' and some subterm g of the right-hand-side (RHS) of a $\mathcal{G}$-rule.*

▸ Notational Convention 2.8. Henceforth we assume a program $\mathcal{P} = \{\, l_i \to r_i \mid 1 \leq i \leq p \,\}$ and a set I of input terms over a ranked alphabet $\Sigma = \Delta \uplus \Gamma$, where Δ consists of defined-operator symbols and Γ consists of constructor symbols. We further assume that the input $I := \mathit{Reach}^o_{\mathcal{G}_0}$ where $\mathcal{G}_0$ is a (uniform) ILTG with start symbol R_0. We aim to over-approximate the collecting semantics of $\mathcal{P}$ on I by means of ILTGs, ranged over by $\mathcal{G}$, that are defined over a terminal alphabet which is set to be Σ, and a non-terminal alphabet $\mathcal{N}$ that satisfies

$$\mathcal{N} \supseteq \{\, X \mid X \in \mathit{Vars}(l_i), 1 \leq i \leq p \,\} \cup \{\, R_0, R_1, \ldots, R_p \,\}$$

where R_0 is the start symbol. Note that every symbol of $\mathcal{P}$ (whether user-defined or constructor) is a terminal symbol of $\mathcal{G}$; and every program variable of $\mathcal{P}$ is a non-terminal of $\mathcal{G}$. For each $1 \leq i \leq p$, the non-terminal R_i (read "the results of the rule $l_i \to r_i$") is intended to generate a superset of all the terms that are reachable from l_i in a rewriting sequence that originates from a term in I.

We make precise what it means for an ILTG to be a safe over-approximation of the collecting semantics of $\mathcal{P}$ on I, and distinguish two versions of safety.

▸ **Definition 2.9** (Safety)**.** Let $(\Sigma, \mathcal{N}, \mathcal{F}, S, \mathcal{G})$ be an ILTG.

(i) $\mathcal{G}$ is *globally safe* for $\mathcal{P}$ on I just if there are terms (i.e. elements of $\mathcal{T}_\Sigma(\mathcal{N}\mathcal{F}^*)$)

- $\widetilde{R_0}, \widetilde{R_1}, \ldots, \widetilde{R_p}$, and
- $\widetilde{X}$, for each $X \in \mathit{Vars}(r_i)$, each $1 \leq i \leq p$

such that for every i and $(\sigma, g) \in Z_i$, $\widetilde{R_i} \to^*_{\mathcal{G}} g$; and $\widetilde{X} \to^*_{\mathcal{G}} \sigma X$ for each $X \in \mathit{Vars}(r_i)$.

(ii) $\mathcal{G}$ is *locally safe* for $\mathcal{P}$ on I just if for every i and $(\sigma, g) \in Z_i$, there is an index sequence δ_σ such that $R_i\,\delta_\sigma \to^*_{\mathcal{G}} g$; and $X\,\delta_\sigma \to^*_{\mathcal{G}} \sigma X$ for each $X \in \mathit{Vars}(r_i)$.

Note that regular tree grammars are ILTGs with an empty index set. Both versions of safety subsume Jones and Andersen's (which assumes regularity of both the input and the approximating grammar).

3 The Grammar Completion Algorithm

Suppose we want to investigate the effect the program $\mathcal{P}$ in Example 2.1 has on the set $Reach_{\mathcal{G}}$ where the ILTG $\mathcal{G}$ is defined in Example 3.1.

▸ **Example 3.1.** $\mathcal{N} = \{\mathsf{R}_0,\mathsf{R}_1,\mathsf{R}_1^{\sigma_1},\mathsf{R}_1^{\sigma_2},\mathsf{R}_2,\mathsf{R}_2^{\sigma_1},\mathsf{R}_2^{\sigma_2}\}$, $\mathcal{F} = \{\sigma_1,\sigma_2\}$, $\Sigma_0 = \{0\}$, $\Sigma_1 = \{\mathsf{counter},\mathsf{genh},\mathsf{genk},\mathsf{S},\mathsf{h},\mathsf{k}\}$, $\Sigma_2 = \{\mathsf{f}\}$

$$\begin{array}{ll}
\mathsf{R}_0 \to \mathsf{counter}(0) \mid \mathsf{R}_1^{\sigma_1} \mid \mathsf{R}_2^{\sigma_1} & \mathsf{R}_1 \to \mathsf{counter}(\mathsf{S}(\mathsf{X})) \mid \mathsf{R}_1^{\sigma_2} \mid \mathsf{R}_2^{\sigma_2} \\
\mathsf{R}_1^{\sigma_1} \to \mathsf{R}_1\sigma_1 & \mathsf{R}_2 \to \mathsf{f}(\mathsf{genh}(\mathsf{X}),\mathsf{genk}(\mathsf{X})) \\
\mathsf{R}_1^{\sigma_2} \to \mathsf{R}_1\sigma_2 & \mathsf{X}\sigma_1 \to 0 \\
\mathsf{R}_2^{\sigma_1} \to \mathsf{R}_2\sigma_1 & \mathsf{X}\sigma_2 \to \mathsf{S}(\mathsf{X}) \\
\mathsf{R}_2^{\sigma_2} \to \mathsf{R}_2\sigma_2 &
\end{array}$$

If we rewrite the term $\mathsf{R}_2\sigma_2\sigma_1$ (which is easily seen to be in $Reach_{\mathcal{G}}$)

$$\mathsf{R}_2\sigma_2\sigma_1 \to_{\mathcal{G}}^* \mathsf{f}(\mathsf{genh}(\mathsf{S}(\mathsf{X}\sigma_1)),\mathsf{genk}(\mathsf{X}\sigma_2\sigma_1)) \to_{\mathcal{P}} \mathsf{f}(\mathsf{h}(\mathsf{genh}(\mathsf{X}\sigma_1)),\mathsf{genk}(\mathsf{X}\sigma_2\sigma_1)) =: t$$

then we observe that t is $\mathcal{P}$ reachable from a term in $Reach_{\mathcal{G}}$, but t is not in $Reach_{\mathcal{G}}$ which thus can be seen not to be invariant under $\mathcal{P}$. However, if we place the rules of Example 3.4 and $\mathcal{G}$ in a new ILTG $\mathcal{G}'$ then using the latter we can rewrite

$$\mathsf{R}_2\sigma_2\sigma_1 \to^* \mathsf{f}(\mathsf{R}_4\sigma_3\sigma_1),\mathsf{genk}(\mathsf{X}\sigma_2\sigma_1)) \to^* \mathsf{f}(\mathsf{h}(\mathsf{genh}(\mathsf{X}\sigma_1)),\mathsf{genk}(\mathsf{X}\sigma_2\sigma_1)).$$

Thus $\mathcal{G}'$ can be seen as a partial completion of $\mathcal{G}$ with respect to $\mathcal{P}$; the former is however still not complete w.r.t. $\mathcal{P}$. In fact, $\mathcal{G}$ is an intermediate result of our algorithm to build an over-approximation of $\mathcal{P}$ on the input $Reach_{\mathcal{G}_0}$ where $\mathcal{G}_0$ is the ILTG with a single rule $\mathsf{R}_0 \to \mathsf{counter}(0)$ and start symbol R_0 .

We aim to over-approximate the collecting semantics of $\mathcal{P}$ on I by means of ILTGs $\mathcal{G}$ that conform to Notational Convention 2.8, using an operator on ITLG, $\delta^n_{\mathcal{P}}(\text{-})$, which we will introduce shortly. First we define an auxiliary operation that takes a rule of an ILTG and returns a set of rules.

▸ **Definition 3.2** (Ext-Operator)**.** Let $\mathcal{G}$ be an ILTG, $n \geq 0$ and $A\gamma \to C[\mathrm{dist}_{\alpha_1\ldots\alpha_k}(g')] \in \mathcal{G}$ where $g' \in \mathcal{T}_\Sigma(\mathcal{N})$. The set $\mathrm{Ext}^n_{\mathcal{P},\mathcal{G}}(A\gamma \to C[\mathrm{dist}_{\alpha_1\ldots\alpha_k}(g')])$ contains (only) the following rules

$$\begin{cases}
A\gamma \to C[R_i^\sigma \alpha_1 \ldots \alpha_k] & \text{(I)} \\
R_i^\sigma \beta_1 \ldots \beta_{\min(m,n)} \to \begin{cases} R_i\,\sigma & \text{if } m \leq n \\ R_i\,\sigma\,\top & \text{otherwise} \end{cases} & \text{(II)} \\
R_i \to r_i & \text{(III)} \\
X\sigma \to \mathrm{aprx}_n(\sigma_0 X), \text{for each } X \in \mathit{Vars}(r_i) & \text{(IV)}
\end{cases}$$

whenever there exist an index sequence $\overline{\beta} = \beta_1 \ldots \beta_m$, a substitution σ_0, and an $1 \leq i \leq n$ such that

(1) $\overline{\beta}$ and $\overline{\alpha} = \alpha_1 \ldots \alpha_k$ are compatible sequences (i.e. one is a prefix of the other), and
(2) σ_0 is a minimally reachable match of $\mathrm{dist}_{\beta_1\ldots\beta_m}(g')$ against l_i, and
(3) σ is the index (*qua* substitution) defined by $\sigma X := \mathit{erase}(\sigma_0 X)$ for each variable X, where $\mathit{erase}(t)$ erases every index sequence, and every superscript of every non-terminal, that occur in $t \in \mathcal{T}_\Sigma(\mathcal{N}\mathcal{F}^*)$; for example

$$\mathit{erase}(g(f(R_1^\tau \alpha_1\alpha_2, h(a, X\beta_1\beta_2\beta_3)))) = g(f(R_1, h(a, X)))$$

$$\text{where } \mathrm{aprx}_i(A\,\alpha_1 \dots \alpha_k) := \begin{cases} A\,\top & \text{if } i = 0 \\ A\,\alpha_1 \dots \alpha_k & \text{if } i > 0 \wedge k \le i \\ A\,\alpha_1 \dots \alpha_i\,\top & \text{if } i > 0 \wedge k > i. \end{cases}$$

The function aprx_i is extended to terms and substitutions in the natural way.

▸ Remark 3.3. (i) In rule of type (IV) above, X on the LHS is a non-terminal; the expression $\mathrm{aprx}_n(\sigma_0 X)$ on the RHS denotes the term obtained by applying the *substitution* $\mathrm{aprx}_n(\sigma_0)$ to the *variable* X (of $\mathcal{P}$). (ii) Several minimally reachable matches σ_0 may give rise to the same substitution σ. The equation in condition (3) is intended to "merge" indices so that only a finite number of indices is eventually generated. (iii) In case $m > n$, the type-(II) rule is added: $R_i^\sigma \alpha_1 \dots \alpha_n \to R_i\,\sigma\,\top$. The operation $\mathrm{Ext}^n_{\mathcal{P},\mathcal{G}}(\text{-})$ does not produce a rule with $R_i\,\sigma$ or $R_i\,\sigma\,\top$ on the LHS. The point of the distinguished index $\top$—as indicated in the definition of $\top(\mathcal{G})$ in the following—is to introduce a measure of non-determinacy, conflating all possible instantiations of a variable X by substitutions introduced during the construction. (iv) Every rule of $\mathrm{Ext}^n_{\mathcal{P},\mathcal{G}}(A\,\gamma \to C[\mathrm{dist}_{\alpha_1 \dots \alpha_k}(g')])$ has index sequences of length at most $n+1$, assuming that $A\,\gamma \to C[\mathrm{dist}_{\alpha_1 \dots \alpha_k}(g')]$ itself has index sequences of length at most n.

▸ **Example 3.4.** The following are the rules in $\mathrm{Ext}^1_{\mathcal{G},\mathcal{P}}(\mathsf{R}_2{\to}\mathsf{f}([\mathsf{genh}(\mathsf{X})],\mathsf{genk}(\mathsf{X})))$ where $\mathcal{G}$ and $\mathcal{P}$ are as before.

$$\begin{array}{lll} \mathsf{R}_2 \to \mathsf{f}(\mathsf{R}_4^{\sigma_3},\mathsf{genk}(\mathsf{X})) & \mathsf{R}_3^{\sigma_4}\sigma_1 \to \mathsf{R}_3\sigma_4 & \mathsf{R}_4^{\sigma_3}\sigma_2 \to \mathsf{R}_4\sigma_3 \\ \mathsf{R}_2 \to \mathsf{f}(\mathsf{R}_3^{\sigma_4},\mathsf{genk}(\mathsf{X})) & \mathsf{R}_3 \to \mathsf{a} & \mathsf{R}_4 \to \mathsf{h}(\mathsf{genh}(\mathsf{X})) \\ & \mathsf{X}\sigma_3 \to \mathsf{X} & \end{array}$$

The rules are the result of the two minimally reachable matches σ_3 (which is the substitution $\{\mathsf{X} \mapsto \mathsf{X}\}$) of $\mathsf{genh}(\mathsf{X}\sigma_2)$ against $\mathsf{genh}(\mathsf{S}(\mathsf{X}))$ and σ_4 (which is the empty substitution) of $\mathsf{genh}(\mathsf{X}\sigma_1)$ against $\mathsf{genh}(\mathsf{0})$. The two patterns are the LHSs of $\mathcal{P}$'s 4th and 3rd rule.

An ILTG completion algorithm Let $n \ge 0$. Using the operation $\mathrm{Ext}^n_{\mathcal{P},\mathcal{G}}(\text{-})$ on $\mathcal{G}$-rules, we first define an operator $\delta^n_{\mathcal{P}}(\text{-})$ on ILTGs, and then construct a sequence of ILTGs by iterating it.

$$\delta^n_{\mathcal{P}}(\mathcal{G}) := \bigcup_{A\,\gamma \to C[\mathrm{dist}_{\alpha_1 \dots \alpha_k}(g')] \in \mathcal{G}} \mathrm{Ext}^n_{\mathcal{P},\mathcal{G}}(A\,\gamma \to C[\mathrm{dist}_{\alpha_1 \dots \alpha_k}(g')]) \;\cup\; \top(\mathcal{G})$$

where (writing $\gamma' < \gamma$ to mean "γ' is a proper prefix of γ")

$$\top(\mathcal{G}) := \bigcup_{\substack{A \ne R_i^\sigma \\ A\gamma \to t \in \mathcal{G}}} \bigcup_{\gamma' < \gamma} \{A\gamma'\top \to \mathrm{aprx}_0(t)\} \;\cup \bigcup_{R_i^\sigma \gamma \to t \in \mathcal{G}} \bigcup_{\gamma' < \gamma} \{R_i^\sigma \gamma'\top \to \mathrm{aprx}_1(t)\}$$

The construction then proceeds as follows. Starting off with an ILTG $\mathcal{G}_0$, we inductively construct a sequence of ILTGs by $\mathcal{G}^n_0 := \mathcal{G}_0$ and $\mathcal{G}^n_{i+1} := \mathcal{G}^n_i \cup \delta^n_{\mathcal{P}}(\mathcal{G}^n_i)$. For each $n \ge 0$, this gives an increasing sequence of ILTGs (*qua* sets of rules): $\mathcal{G}^n_0 \subseteq \mathcal{G}^n_1 \subseteq \mathcal{G}^n_2 \subseteq \mathcal{G}^n_3 \subseteq \dots$

▸ Remark 3.5. The seed, $\mathcal{G}_0$, is an arbitrary uniform ILTG (conforming to Notational Convention 2.8). Note that uniformity is not a real restriction since every ILTG can be transformed to an equivalent uniform ILTG. It follows from Proposition 2.4 that the input set of our algorithm can be an arbitrary pushdown-tree language.

▸ **Example 3.6.** Combining the rules below and Examples 3.1 and 3.4 we get the result of our procedure to approximate the collecting semantics of the program $\mathcal{P}$ in Example 2.1 on the input grammar $\mathcal{G}_0 = \{\mathsf{R}_0 \to \mathsf{counter}(\mathsf{0})\}$.

//... the rules of Examples 3.1 and 3.4
$\mathsf{R}_2 \to \mathsf{f}(\mathsf{R}_4^{\sigma_3}, \mathsf{R}_5^{\sigma_4}) \mid \mathsf{f}(\mathsf{R}_4^{\sigma_3}, \mathsf{R}_6^{\sigma_3})$
$\mathsf{R}_2 \to \mathsf{f}(\mathsf{R}_3^{\sigma_4}, \mathsf{R}_5^{\sigma_4}) \mid \mathsf{f}(\mathsf{R}_3^{\sigma_4}, \mathsf{R}_6^{\sigma_3})$
$\mathsf{R}_4 \to \mathsf{h}(\mathsf{R}_3^{\sigma_4}) \mid \mathsf{h}(\mathsf{R}_4^{\sigma_3})$
$\mathsf{R}_5^{\sigma_4}\sigma_1 \to \mathsf{R}_5\sigma_4$
$\mathsf{R}_6^{\sigma_3}\sigma_2 \to \mathsf{R}_6\sigma_3$
$\mathsf{R}_5 \to \mathsf{b}$
$\mathsf{R}_6 \to \mathsf{k}(\mathsf{genk}(\mathsf{X})) \mid \mathsf{k}(\mathsf{R}_5^{\sigma_4}) \mid \mathsf{k}(\mathsf{R}_6^{\sigma_3})$

Note that the results introduced by the operator $\mathsf{T}(-)$ have been omitted. It can be easily seen that the reachable ground terms of the resulting ILTG are precisely the terms reachable by $\mathcal{P}$ from $\mathcal{G}_0$. Further the ground terms reachable from $\mathsf{R}_2\sigma_2^n\sigma_1$ are

$$\mathsf{f}(\mathsf{h}^i(\mathsf{genh}^{n-i}(\mathsf{a}),\mathsf{k}^j(\mathsf{genk}^{n-j}(\mathsf{b}))))$$

which are the result terms bound to the substitution $\sigma_2^n \circ \sigma_1$ in Z_2 in $\mathcal{P}$'s collecting semantics (where we write $f^n = f \circ \ldots \circ f$ n-times), letting n range over all n we get precisely Z_2.[5] Moreover, the set of reachable ground constructor terms from R_0 is precisely $\{\, f(h^n(a), k^n(b)) \mid n \geq 0 \,\}$. Note that the tree language generated by $\mathcal{P}$ from $\mathcal{G}_0$ is non-regular. Thus, we can see that our algorithm makes use of the greater expressivity provided by ILTGs to describe this set.

4 Termination

There are two key ideas in the termination proof of the ILTG completion algorithm. The first, due to Jones and Andersen, concerns minimally reachable match. By considering only these substitutions when constructing $\mathrm{Ext}^n_{\mathcal{P},\mathcal{G}_i}(\text{-})$, the RHS of every rule that is generated is guaranteed to be a *variant* of a subterm of the RHS of either a $\mathcal{P}$-rule or a $\mathcal{G}_0$-rule; and the set of such variants is bounded. Secondly, the merging of substitutions via *erase* ensures that only finitely many indices are generated eventually.

Variants Let $\mathcal{X}$ be a set of terms. A $\mathcal{X}$-*variant* of t is a term obtained from t by replacing one or more subterms by an element of $\mathcal{X}$. For example, the term $f(g(A\alpha\beta, B\alpha\alpha\alpha), a)$ is a $\mathcal{N}'\mathcal{F}'^{\leq 3}$-variant of the term $t = f(g(h(A), f(A', b, c), a)$, where $\mathcal{N}' = \{A, B\}$ and $\mathcal{F}' = \{\alpha, \beta\}$ and we write $\mathcal{A}^{\leq k}$ for the set of sequences of elements of $\mathcal{A}$ of length no more than k. Note that the term t trivially is a $\mathcal{T}_\Sigma(\mathcal{N}\mathcal{F}^*)$ -variant of itself. The following definition makes the variant relation precise.

▸ **Definition 4.1** ($\mathcal{X}$-Variant)**.** Let $\mathcal{X} \subseteq \mathcal{T}_\Sigma(\mathcal{N}\mathcal{F}^*)$. We define the $\mathcal{X}$-variant relation $\sqsubseteq_\mathcal{X} \subseteq \mathcal{T}_\Sigma(\mathcal{N}\mathcal{F}^*) \times \mathcal{T}_\Sigma(\mathcal{N}\mathcal{F}^*)$ by induction over the following rules:

- $t \sqsubseteq_\mathcal{X} t$ for $t \in \mathcal{T}_\Sigma(\mathcal{N}\mathcal{F}^*)$
- $s \sqsubseteq_\mathcal{X} t$ for $s \in \mathcal{X}$ and $t \in \mathcal{T}_\Sigma(\mathcal{N}\mathcal{F}^*)$
- if $t_i \sqsubseteq_\mathcal{X} t'_i$ for each $1 \leq i \leq n$ then $f(t_1, \ldots, t_n) \sqsubseteq_\mathcal{X} f(t'_1, \ldots, t'_n)$

If $s \sqsubseteq_\mathcal{X} t$ we say s is an $\mathcal{X}$-*variant* of t.

We note at this point that for a fixed $\mathcal{X}$ the relation $\sqsubseteq_\mathcal{X}$ is reflexive and transitive.

The next proposition captures the observation that our flow analysis procedure does not really add "new information". It turns out that all RHSs of rules in $\mathcal{G}_i^n$ are in fact variants of subterms of terms t where t ranges over the RHSs of rules in $\mathcal{G}_0$ and $\mathcal{P}$.

For $i, n \geq 0$, define $\mathcal{F}_{i,n}$ be the index set of $\mathcal{G}_i^n$ and write $\mathcal{N}_{i,n}$ ($\mathcal{N}_0$) for the set of non-terminals occurring in $\mathcal{G}_i^n$ ($\mathcal{G}_0$).

[5] Owing to the "merging" of substitutions, it is not always possible to identify indices with substitutions.

▸ **Proposition 4.2.** *Assume that $\mathcal{G}_0$ is uniform. For each $i, n \geq 0$ and each rule $A\,\gamma \to t$ in $\mathcal{G}_i^n$, there exist a rule $l \to r$ in $\mathcal{P} \cup \mathcal{G}_0$ and $r' \leq r$ such that $t \sqsubseteq_{\mathcal{X}} r'$ where $\mathcal{X} = \mathcal{N}_{i,n}\mathcal{F}_{i,n}^{\leq \max(n+1,2)}$, and t is uniform.*

In the following we write $\mathcal{V}_{\mathcal{P}} = \bigcup_{i=1}^{p} Vars(l_i)$ and $\mathcal{N}_{\mathcal{P}} = \{ R_i \mid 1 \leq i \leq p \}$.

▸ **Lemma 4.3.** *For each $i, n \geq 0$, $\mathcal{F}_{i,n} \subseteq \mathcal{F}_0 \cup \{ \top \} \cup \{ \sigma \mid \sigma$ has type $\mathcal{V}_{\mathcal{P}} \to \mathcal{Y} \}$ where*

$$\mathcal{Y} \coloneqq \{ t \in \mathcal{T}_\Sigma(\mathcal{N}\mathcal{F}^*) \mid \text{there exist } l \to r \in \mathcal{G}_0 \cup \mathcal{P} \text{ and } r' \leq r \text{ such that } t \sqsubseteq_{\mathcal{N}_{\mathcal{P}} \cup \mathcal{V}_{\mathcal{P}}} r' \}$$

and $\mathcal{F}_0$ is the index set of $\mathcal{G}_0$ (which is defined to be $\varnothing$ in case $\mathcal{G}_0$ is regular). Hence there is some $m \geq 0$ such that for all $i, n \geq 0$, $|\mathcal{F}_{i,n}| \leq m$.

Termination of our completion procedure is an immediate consequence of Proposition 4.2 and Lemma 4.3.

▸ **Theorem 4.4** (Termination)**.** *For each $n \geq 0$, there is some $i \geq 0$ such that $\mathcal{G}_i^n = \mathcal{G}_{i+1}^n$. I.e. the algorithm terminates.*

We will refer to the fixpoint ILTG by $\mathcal{G}^n$ from now on.

We can give the following size bound for $\mathcal{G}^n$.

$$|\mathcal{G}^n| = O\left((\text{size}(\mathcal{P}) + \text{size}(\mathcal{G}_0))^{2|\mathcal{V}_{\mathcal{P}}| m(n,\mathcal{D})} \times \text{size}(\mathcal{P})^{\mathcal{D}|\mathcal{V}_{\mathcal{P}}| m(n,\mathcal{D}) + \mathcal{D} + 1} \right)$$

where $m(n, \mathcal{D}) = \mathcal{D}(\max(n+1, 2) + 1) + n + 1$, $\text{size}(\mathcal{P})$ is linear in $|\mathcal{P}|$, $|\mathcal{V}_{\mathcal{P}}|$ and the number of subterms of the largest RHS of $\mathcal{P}$, and $\text{size}(\mathcal{G}_0)$ is linear in $|\mathcal{G}_0|$, the number of $\mathcal{G}_0's$ non-terminals, indices and LHSs and number of subterms of the largest RHS of $\mathcal{G}_0$. Note that $\mathcal{D}$ is greatest number $(k+1)^d$ such that $d = \text{depth}(r)$ and k is the arity of a Σ symbol occurring in r, where r ranges over the RHSs of $\mathcal{P}$ and $\mathcal{G}_0$. Thus, we can see the size of $\mathcal{G}^n$ is polynomial in the number of rules in $\mathcal{P}$ and $\mathcal{G}_0$, and exponential in n, $\mathcal{D}$ and the number of variables.[6] For comparison, a similar analysis yields that the size of the fixpoint grammar of Jones and Andersen's procedure is $O(\text{size}(\mathcal{G}_0) + \text{size}(\mathcal{P}))^3 \times \text{size}(\mathcal{P})^{\mathcal{D}})$.

5 Soundness: Local and Global Safety

A program $\mathcal{P}$ (or an ILTG $\mathcal{G}$) determines a transition graph whose vertices are terms and whose edge-set is the rewrite relation of the program (or grammar). In such a setting it is natural to consider *simulation*. A key insight of our soundness proof is that reachability under ILTG $\mathcal{G}^n$ is *invariant under $\mathcal{P}$-transition, modulo simulation*. I.e. whenever a term $t \in \mathcal{T}_\Sigma(\mathcal{N}\mathcal{F}^*)$, which is reachable from s under rewriting by $\mathcal{G}^n$, can make a $\mathcal{P}$-transition to t', then there is a term, which is reachable from s under rewriting by $\mathcal{G}^n$, that simulates t'.

The main technical result (Theorem 5.12) is that $\mathcal{G}^n$ is locally safe, from which global safety follows. We organise our proof as follows. First, we identify a crucial property of ILTGs that are candidates for approximating the collecting semantics of $\mathcal{P}$ on I, called *emulation* (Definition 5.5). We show that emulation implies invariance under $\mathcal{P}$-transition, modulo simulation (Proposition 5.6), which implies local safety (Proposition 5.7). It then remains to show that $\mathcal{G}^n$ satisfies emulation (Theorem 5.10).

[6] Note that $\mathcal{D}$ can be made into a constant by enforcing a constraint on the depth of RHSs. A program can be transformed into a conforming program by introducing "subroutines" to reduce the depth of RHSs. The increase in rules is then polynomial in the length of $\mathcal{P}$.

Simulation Fix $\Sigma, \mathcal{N}, \mathcal{F}$ and a program $\mathcal{P} = \{ l_i \to r_i \mid 1 \leq i \leq p \}$ (over Σ) and let the meta-variable $\mathcal{R}$ range over $\mathcal{P}$ and ILTGs $\mathcal{G}$.

▸ **Definition 5.1** (Σ-equal $\mathcal{R}$-simulation)**.** We call a relation $R \subseteq (\mathcal{T}_\Sigma(\mathcal{N}\mathcal{F}^*))^2$ a Σ-*equal pre-$\mathcal{R}$-simulation* just if $R \subseteq F_\mathcal{R}(R)$, where $F_\mathcal{R}(R)$ is defined to be the set of pairs $(t_1, t_2) \in (\mathcal{T}_\Sigma(\mathcal{N}\mathcal{F}^*))^2$ satisfying

(i) $\widehat{t_1} = \widehat{t_2}$, where $\widehat{t} \in \mathcal{T}_\Sigma(*)$ is obtained from t by replacing every $\mathcal{N}\mathcal{F}^*$-subterm of t by a special symbol $*$, which is assumed not to be a member of $\Sigma \cup \Delta$; and
(ii) for every t_1' if $t_1 \to_\mathcal{R} t_1'$ then there exists t_2' such that $t_2 \to_\mathcal{R} t_2'$ and $(t_1', t_2') \in R$.

Since $F_\mathcal{R}$ is a monotone function, by Knaster-Tarski Fixpoint Theorem, it has a greatest fixpoint, which we write as $\preceq_\mathcal{R}$ and refer to as Σ-*equal $\mathcal{R}$-simulation* (or simply *$\mathcal{R}$-simulation*). We read $s \preceq_\mathcal{R} t$ as "t $\mathcal{R}$-simulates s".

We denote the intersection of the $\mathcal{G}$-simulation and $\mathcal{P}$-simulation by $\preceq_{\mathcal{G},\mathcal{P}}$ i.e. $\preceq_{\mathcal{G},\mathcal{P}} := \preceq_\mathcal{G} \cap \preceq_\mathcal{P}$. Whenever it is clear from the context, we drop the subscript and write it simply as $\preceq$.

▸ **Proposition 5.2.** *Let $\mathcal{G}$ be an ILTG, then the relation $\preceq$ has the following properties.*

(i) Whenever two terms are related by $\preceq$, if one of them is ground (i.e. a member of $\mathcal{T}_\Sigma$) then so is the other, and they are equal terms.
(ii) $\preceq$ is a pre-congruence i.e. if $t \preceq t'$ then $C[t] \preceq C[t']$.
(iii) $\preceq$ is a preorder on $\mathcal{T}_\Sigma(\mathcal{N}\mathcal{F}^*)$.
(iv) If $t_1 \preceq t_2$ and $t_1 \to^* t_1'$ then there is a term t_2' such that $t_2 \to^* t_2'$ and $t_1' \preceq t_2'$
(v) If $t \preceq t'$ and $t \to^* s$ for some $s \in \mathcal{T}_\Sigma$, then $t' \to^* s$.
(vi) If $t \preceq t'$ then $Reach^o_\mathcal{G}(t) \subseteq Reach^o_\mathcal{G}(t')$.

▸ **Proposition 5.3.** *Let t_0 be a term such that if $t \in Reach_\mathcal{G}(t_0)$ and $t \to_\mathcal{P} t'$ then there exists $t'' \in Reach_\mathcal{G}(t_0)$ such that $t' \preceq t''$. Then*

(i) for each $n \geq 0$, if $t \in Reach_\mathcal{G}(t_0)$ and $t \to^n_\mathcal{P} t'$ then there exists $t'' \in Reach_\mathcal{G}(t_0)$ such that $t' \preceq t''$.
(ii) $Reach^o_\mathcal{P}(Reach_\mathcal{G}(t_0)) \subseteq Reach^o_\mathcal{G}(t_0)$

Emulation The collecting semantics of a program $\mathcal{P}$ on an input set I can be characterised as follows [10, Lemma 2.6]. We aim to introduce a notion (called emulation) that mimicks the property.

▸ **Lemma 5.4** (Jones and Andersen 2007)**.** *The collecting semantics of $\mathcal{P}$ on I is the smallest (ordered point-wise) tuple of sets of pairs, $(Z_0, Z_1, \ldots, Z_p)$, that satisfies:*

(1) If $g \in I$ then $(id, g) \in Z_0$.
(2) If $(\sigma, C[\sigma' \, l_i]) \in Z_j$ then $(\sigma', \sigma' \, r_i) \in Z_i$ for $0 \leq i, j \leq p$.
(3) If $(\sigma, C[\sigma' \, l_i]) \in Z_j$ and $(\sigma', g') \in Z_i$, then $(\sigma, C[g']) \in Z_j$ for $0 \leq i, j \leq p$.

In the definition to follow, we assume that ITLGs $\mathcal{G}_0$ and $\mathcal{G}$ that satisfy the Notational Convention 2.8, with the input to $\mathcal{P}$ set to $Reach_{\mathcal{G}_0}$. Let the meta-variable A range over the following subset of non-terminals (of $\mathcal{G}_0$ and $\mathcal{G}$)

$$\{ R_0, R_1, \ldots, R_p \} \cup \{ X \mid X \in Vars(r_i), 1 \leq i \leq p \}. \tag{1}$$

▸ **Definition 5.5** (Emulation)**.** An ILTG $\mathcal{G}$ *emulates* the collecting semantics of $\mathcal{P}$ on input $Reach^o_{\mathcal{G}_0}$ just if

(i) $Reach_{\mathcal{G}_0} \subseteq Reach_{\mathcal{G}}(R_0)$
(ii) whenever $C[\sigma\, l_i] \in Reach_{\mathcal{G}}(A\,\gamma)$ then there are $\delta_\sigma \in \mathcal{F}^*$ and substitution σ' such that
  (a) $\sigma'\, r_i \in Reach_{\mathcal{G}}(R_i\,\delta_\sigma)$, $\sigma' Z \in Reach_{\mathcal{G}}(Z\,\delta_\sigma)$ for each $Z \in Vars(r_i)$, and $\sigma \preceq \sigma'$ (i.e. $\sigma\, X \preceq \sigma'\, X$ for all $X \in \mathcal{V}$)
  (b) if $t \in Reach_{\mathcal{G}}(R_i\,\delta_\sigma)$ then $C[t] \in Reach_{\mathcal{G}}(A\,\gamma)$.

▸ **Proposition 5.6.** *Let $\mathcal{G}$ be an ILTG that emulates the collecting semantics of $\mathcal{P}$ on $Reach^o_{\mathcal{G}_0}$. Then, with A ranging over the set (1) of non-terminals, and δ ranging over $\mathcal{F}^*$*

(i) if $t \in Reach_{\mathcal{G}}(A\,\delta)$ and $t \to^*_{\mathcal{P}} t'$ then there exists $t'' \in Reach_{\mathcal{G}}(A\,\delta)$ such that $t' \preceq t''$.
(ii) $Reach^o_{\mathcal{P}}(Reach_{\mathcal{G}}(A\delta)) \subseteq Reach^o_{\mathcal{G}}(A\delta)$.

Proof. (i) Let $t \in Reach_{\mathcal{G}}(A\delta)$. We assume that $t \to_{\mathcal{P}} t'$; the general case of $t \to^*_{\mathcal{P}} t'$ then follows from Proposition 5.3. By assumption, $t = C[\sigma l_i]$ and $t' = C[\sigma r_i]$ for some i. Thus by assumption of emulation there exist σ' and δ_σ such that $\sigma' r_i \in Reach_{\mathcal{G}}(R_i\delta_\sigma)$ and $\sigma \preceq \sigma'$. So $C[\sigma r_i] \preceq C[\sigma' r_i]$ as $\preceq$ is a pre-congruence. Also since $\sigma' r_i \in Reach_{\mathcal{G}}(R_i\delta_\sigma)$, it follows from the emulation assumption that $C[\sigma' r_i] \in Reach_{\mathcal{G}}(A\delta)$.

(ii) Follows from Proposition 5.3 when combined with (i).

◂

▸ **Proposition 5.7.** *Let $\mathcal{G}$ be an ILTG that emulates the collecting semantics of $\mathcal{P}$ on $Reach^o_{\mathcal{G}_0}$. Then $\mathcal{G}$ is locally safe for $\mathcal{P}$ on $Reach^o_{\mathcal{G}_0}$.*

Proof. Let $(\sigma, g) \in Z_i$, then there exists $w \in Reach^o_{\mathcal{G}_0}$ and context $C[\,\text{-}\,]$ such that $w \to^*_{\mathcal{P}} C[\sigma\, l_i]$ and $\sigma\, r_i \to^*_{\mathcal{P}} g$. Note that $C[\sigma\, l_i]$, $\sigma\, r_i$ and g are elements of $\mathcal{T}_\Sigma$. Thus $C[\sigma l_i] \in Reach^o_{\mathcal{P}}(Reach_{\mathcal{G}_0}) \subseteq Reach^o_{\mathcal{P}}(Reach_{\mathcal{G}}(R_0)) \subseteq Reach^o_{\mathcal{G}}(R_0)$, the second inclusion follows from Proposition 5.6(ii). Thus by definition there exist σ' and δ_σ such that $\sigma'\, r_i \in Reach_{\mathcal{G}}(R_i\,\delta_\sigma)$, $\sigma'\, X \in Reach_{\mathcal{G}}(X\,\delta_\sigma)$ for each $X \in Vars(r_i)$, and $\sigma \preceq \sigma'$.

Since $\mathrm{range}(\sigma) \subseteq \mathcal{T}_\Sigma$, $\sigma \preceq \sigma'$ and $\preceq$ is Σ-equal, we have that $\sigma = \sigma'$. Hence we can conclude that in fact $\sigma\, r_i \in Reach_{\mathcal{G}}(R_i\,\delta_\sigma)$ and $\sigma X \in Reach_{\mathcal{G}}(X\,\delta_\sigma)$. Now since $\sigma\, r_i \to^*_{\mathcal{P}} g$, it follows from Proposition 5.6(ii) that $g \in Reach^o_{\mathcal{P}}(Reach_{\mathcal{G}}(R_i\,\delta_\sigma)) \subseteq Reach^o_{\mathcal{G}}(R_i\,\delta_\sigma)$. Thus $R_i\,\delta_\sigma \to^* g$ and $X\,\delta_\sigma \to^* \sigma X$.

◂

▸ Remark 5.8. Define a simulation relation over index sequences by $\gamma \preceq \gamma'$ just if $A\,\gamma \preceq A\,\gamma'$ for all $A \in \mathcal{N}$. If an ILTG is locally safe for $\mathcal{P}$ on $Reach^o_{\mathcal{G}_0}$ and has an index $\top$ (say) that simulates every index (i.e. $\alpha \preceq \top$ for all $\alpha \in \mathcal{F}$), then it is also globally safe, for we can set $\widetilde{R_i} := R_i\,\top$ and $\widetilde{X} := X\,\top$.

$\mathcal{G}^n$ emulates the collecting semantics For our soundness argument it remains to show that for each n, the fixpoint ILTG $\mathcal{G}^n$ emulates the collecting semantics. We begin by stating a technical lemma.

▸ **Lemma 5.9.** *For each $n \geq 0$, if $A\gamma \to^*_{\mathcal{G}^n} C[\sigma l_i]$ then there are σ_0, σ'_0 and δ such that*

(i) $A\gamma \to^*_{\mathcal{G}^n} C[\sigma_0 l_i]$ and $\sigma_0 r_i \to^*_{\mathcal{G}^n} \sigma r_i$
(ii) $A\gamma \to^*_{\mathcal{G}^n} C[R_i\delta]$
(iii) $X\,\delta \to^*_{\mathcal{G}^n} \sigma'_0 X$, for each $X \in Vars(r_i)$
(iv) $\sigma_0 \preceq \sigma'_0$

Emulation is a straightforward consequence of the above result.

▸ **Theorem 5.10.** *For each $n \geq 0$, $\mathcal{G}^n$ emulates the collecting semantics of $\mathcal{P}$ on $Reach^o_{\mathcal{G}_0}$.*

Proof. We will now show that the conditions (i), (iia) and (iib) of Definition 5.5 hold for $\mathcal{G}^n$. By construction it is the case that $\mathcal{G}^n \supseteq \mathcal{G}_0$, thus $Reach_{\mathcal{G}_0} = Reach_{\mathcal{G}_0}(R_0) \subseteq Reach_{\mathcal{G}^n}(R_0)$. Therefore condition (i) is satisfied. For condition (iia) and (iib), suppose that $C[\sigma l_i] \in Reach_{\mathcal{G}^n}(A\gamma)$. By applying Lemma 5.9 to $A\gamma \to^*_{\mathcal{G}^n} C[\sigma l_i]$, it follows that for some δ_σ,σ_0,σ'_0, we have $A\gamma \to^* C[R_i\delta_\sigma]$, $X\delta_\sigma \to^* \sigma'_0 X$ for every X that occurs in r_i, $\sigma_0 r_i \to^*_{\mathcal{G}^n} \sigma r_i$ and $\sigma_0 \preceq \sigma'_0$. Since $\sigma_0 \preceq \sigma'_0$, $\sigma_0 X \to^*_{\mathcal{G}^n} \sigma X$, there is a term $\sigma' X$ such that $\sigma'_0 X \to^*_{\mathcal{G}^n} \sigma' X$ and $\sigma X \preceq \sigma' X$ for all $X \in Vars(r_i)$. Thus we can infer that $\sigma \preceq \sigma'$ and $\sigma'_0 r_i \to^*_{\mathcal{G}^n} \sigma' r_i$. Hence $X\delta_\sigma \to^*_{\mathcal{G}^n} \sigma'_0 X \to^*_{\mathcal{G}^n} \sigma' X$. Further $R_i\delta_\sigma \to \mathrm{dist}_{\delta_\sigma} r_i \to^* \sigma' r_i$ by rewriting all non-terminals $X \in Vars(r_i)$ to $\sigma' X$. We can thus infer that $\sigma' r_i \in Reach_{\mathcal{G}^n}(R_i\delta_\sigma)$, $\sigma' X \in Reach_{\mathcal{G}^n}(X\delta_\sigma)$ where $\sigma \preceq \sigma'$. Hence condition (iia) holds. For condition (iib) further suppose that $t \in Reach_{\mathcal{G}^n}(R_i\delta_\sigma)$, then since $A\gamma \to^* C[R_i\delta_\sigma]$ and $R_i\delta_\sigma \to^* t$ we can rewrite $A\gamma \to^* C[t]$, i.e. $C[t] \in Reach_{\mathcal{G}^n}(A\gamma)$. ◂

Thus it follows from Proposition 5.7 that $\mathcal{G}^n$ is locally safe for each $n \geq 0$. To prove global safety, we combine the following lemma with Remark 5.8.

▸ **Proposition 5.11.** *For all $\gamma_0, \gamma_1, \gamma_2 \in \mathcal{F}^*$ we have $\gamma_0\gamma_1 \preceq \gamma_0 \top \gamma_2$ in $\mathcal{G}^n$.*

To summarise

▸ **Theorem 5.12.** *For each $n \geq 0$, the fixpoint ILTG $\mathcal{G}^n$ is both locally and globally safe for program $\mathcal{P}$ on input $Reach_{\mathcal{G}_0}$.*

6 Related Work

TRS Reachability Problem There is a line of work in the rewriting community devoted to the construction of $Reach_{\mathcal{R}}(I)$ where I is a regular set and $\mathcal{R}$ is a TRS satisfying various restrictions. This is an interesting problem because for a regular input set I, $Reach_{\mathcal{R}}(I)$ is not necessarily regular, even if $\mathcal{R}$ is a confluent and terminating linear TRS [7].

Based on earlier work by Genet [6], Feuillade et al. [5] proposed a tree-automaton completion algorithm for over-approximating $Reach_{\mathcal{R}}(I)$, for a given (left-linear) TRS $\mathcal{R}$ and a regular input set I. The algorithm constructs a sequence of tree automata and is parametrised by an abstraction function that maps terms to states of the automaton. This method is quite versatile as the completion procedure can be fine-tuned by choosing the appropriate abstraction function. However, the approach is not fully automatic; further not every abstraction function is guaranteed to lead to a fixpoint automaton.

Building on this work, Boichut et al. [2] introduced a semi-algorithm which automatically chooses abstraction functions for the completion procedure. Their aim is to obtain a more conclusive analysis of whether a term t is reachable from a given input set. The abstraction function is refined in order to obtain either a fixpoint automaton which does not accept t or an under-approximation of $Reach_{\mathcal{P}}(I)$ which does include t. However, this approach is not guaranteed to terminate.

Model Checking Functional Programs Kobayashi et al. [11] introduced a type-based model-checking method for an extension of higher-order recursion schemes called *higher-order multi-parameter tree transducers* (HMTT). They gave an algorithm for checking if the tree generated by a given HMTT satisfies a given output specification, provided the input trees conform to a given input specification. It is not easy to compare our work with theirs, but two aspects stand out. First their specifications are restricted to regular tree

languages. Secondly patterns (for matching) in their framework are required to satisfy a rigid type constraint; for example, their method cannot handle our Example 2.1 (unless certain invariants on intermediate data structures are provided by the programmer [21]).

Ong and Ramsay [19] recently introduced *pattern-matching recursion schemes* (PMRS) as an accurate model of computation for functional programs that manipulate algebraic data types. They present a verification method that, given an order-n PMRS $\mathcal{P}$ and an input set I generated by a regular tree grammar, constructs an order-n weak PMRS which over-approximates the set of terms reachable from I under rewriting from $\mathcal{P}$. Their construction uses a binding analysis à la Jones and Andersen to over-approximate only the first-order pattern-matching behaviour, whilst remaining completely faithful to the higher-order control flow. We believe that their binding analysis can be refined by using a variant of our ILTG-based completion algorithm, thus giving a more accurate over-approximation of pattern-matching in their framework.

XML Type Checking The extensible markup language XML is the standard format for exchanging structured data. Central to XML processing is the *type checking problem*: given an input type, an output type and a transformation f, does f transform every input that conforms to the input type to an output that conforms to the output type? Since the XML type checking problem is undecidable, general solutions are necessarily approximate. However, by restricting types and transformations appropriately, type checking can be made decidable. A recent direction [14, 15] considers types given by languages recognisable by finite-state automata, and transformations specified by (versions of) stay macro tree transducers (SMTTs). SMTTs are first-order functional programs that generate output trees by top-down pattern matching its first (tree) argument, while possibly accumulating intermediate results in the other (tree) parameters: they are essentially first-order pattern-matching recursion schemes [19]. It would be interesting to understand the approach based on SMTTs, as it seems likely that there are connections with our work.

7 Evaluation, Conclusion and Further Directions

Evaluation A few remarks by way of comparison with related work. (i) Our algorithm can take an arbitrary pushdown tree language as the input set. To our knowledge, all published over-approximation results for the reachability problem for left-linear TRS assume a regular input set. (ii) For each fixed $n \geq 0$, our completion method is at least as accurate as Jones and Andersen's [10] (because after erasing all references to indices, our method yields the same result as Jones and Andersen's, modulo some superfluous rules). (iii) A source of inaccuracy in Jones and Andersen's approach is the decoupling of the pairing $(\sigma, t) \in Z_i$ (in the collecting semantics) between a reachable substitution σ and the associated result term t. The notion of emulation allows us to introduce a weak form of coupling between σ and t in the sense of local safety. The definition of emulation also applies to Jones and Andersen's fixpoint regular tree grammar. However, in the absence of indices, $\delta_\sigma = \epsilon$. Thus, we can see that the coupling of program results and program states (substitutions) is stronger in our fixpoint ILTG. (iv) It is not straightforward to compare our result with related work from the rewriting community [9, 5, 2]. We can see from Example 2.1 that our approach accurately captures a non-regular set of reachable terms. Because Feuillade et al. and Boichut et al. use finite tree automata, such accuracy is beyond their reach. However, there is an example in the full version of this paper [13] for which our algorithm produces a strict over-approximation which is regular, whereas, by a judicious choice of parameters,

the algorithm of Feuillade et al. can yield a better over-approximation. On the other hand, our algorithm is fully automatic and guaranteed to terminate for each fixed n (as is Jones and Andersen's), neither of which is true of the methods of Feuillade et al. and of Boichut et al..

Conclusion Using indices to capture (sets of) substitutions, we have presented a completion algorithm which, given a left-linear TRS $\mathcal{P}$, an input set described by an ILTG $\mathcal{G}_0$ and $n \geq 0$, constructs an ILTG $\mathcal{G}^n$ that safely over-approximates the collecting semantics of $\mathcal{P}$ on $\mathcal{G}_0$. To our knowledge, this is the first completion procedure for pushdown tree automata, and it yields a strong approximation result for the left-linear TRS reachability problem.

Further Directions (i) A priority is to construct an implementation of our completion algorithm for empirical evaluation. (ii) A key idea of our approach is to merge substitutions, σ and σ', just when $\mathit{erase}(\sigma) = \mathit{erase}(\sigma')$. One way to improve our algorithm is to refine the merge operation. For example, one could define $\mathit{erase}^1(A\alpha_1 \dots \alpha_n) := A\,\mathit{erase}(\alpha_1) \dots \mathit{erase}(\alpha_n)$. A similar argument to our current termination algorithm should apply. One could envisage an inductive definition in the same style for erase^2, erase^3, ... (iii) It would be interesting to identify sufficient conditions for the ILTG completion algorithm to be accurate for reachability i.e. for which class of programs does the fixpoint ILTG generate precisely the set of reachable terms?

Acknowledgements Financial support by EPSRC (research grant EP/F036361/1 and OUCL DTG Account doctoral studentship for the first author) is gratefully acknowledged. We would like to thank Damien Sereni and Steven Ramsay for helpful discussions and insightful comments, and the anonymous reviewers for their detailed reports.

References

1 Alfred V. Aho. Indexed grammars - an extension of context-free grammars. *Journal of the ACM (JACM)*, 15(4):647–671, 1968.

2 Yohan Boichut, Roméo Courbis, Pierre-Cyrille Héam, and Olga Kouchnarenko. Finer is better: Abstraction refinement for rewriting approximations. In *Proceedings of Rewriting Techniques and Applications (RTA)*, volume 5117 of *Lecture Notes in Computer Science*, pages 48–62. Springer, 2008.

3 Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the Symposium on Principles of Programming Languages (POPL)*, pages 238–252. ACM, 1977.

4 Joost Engelfriet and Heiko Vogler. Pushdown machines for the macro tree transducer. *Theoretical Computer Science*, 42:251–368, 1986.

5 Guillaume Feuillade, Thomas Genet, and Valérie Viet Triem Tong. Reachability Analysis over Term Rewriting Systems. *Journal of Automated Reasoning*, 33(3-4):341–383, 2005.

6 Thomas Genet. Decidable approximations of sets of descendants and sets of normal forms. In *Proceedings of Rewriting Techniques and Applications (RTA)*, volume 1379 of *Lecture Notes in Computer Science*, pages 151–165. Springer, 1998.

7 Rémi Gilleron and Sophie Tison. Regular tree languages and rewrite systems. *Fundamenta Informaticae*, 24(1/2):157–174, 1995.

8 Irène Guessarian. Pushdown tree automata. *Mathematical Systems Theory*, 16(4):237–263, 1983.

9 Florent Jacquemard. Decidable approximations of term rewriting systems. In *Proceedings of Rewriting Techniques and Applications (RTA)*, volume 1103 of *Lecture Notes in Computer Science*, pages 362–376. Springer, 1996.

10 Neil D. Jones and Nils Andersen. Flow analysis of lazy higher-order functional programs. *Theoretical Computer Science*, 375(1-3):120–136, 2007.

11 Naoki Kobayashi, Naoshi Tabuchi, and Hiroshi Unno. Higher-order multi-parameter tree transducers and recursion schemes for program verification. In *Proceedings of the Symposium on Principles of Programming Languages (POPL)*, pages 495–508. ACM, 2010.

12 Jonathan Kochems. Approximating reachable terms of functional programs. Oxford University MMathsCS 4th-year Project Report, 2010.

13 Jonathan Kochems and C.-H. Luke Ong. Improved functional flow and reachability analyses using indexed linear tree grammars. Long version, 2010.

14 Sebastian Maneth, Alexandru Berlea, Thomas Perst, and Helmut Seidl. Xml type checking with macro tree transducers. In *Proceedings of the Symposium on Principles of Database Systems (PODS)*, pages 283–294. ACM, 2005.

15 Sebastian Maneth, Thomas Perst, and Helmut Seidl. Exact xml type checking in polynomial time. In *Proceedings of the International Conference on Database Theory (ICDT)*, volume 4353 of *Lecture Notes in Computer Science*, pages 254–268. Springer, 2007.

16 A. N. Maslov. Multilevel stack automata. *Problems of Information Transmission*, 12:38–43, 1976.

17 C.-H. Luke Ong. On model-checking trees generated by higher-order recursion schemes. In *Proceedings of Symposium on Logic in Computer Science (LICS)*, pages 81–90. IEEE Computer Society, 2006.

18 C.-H. Luke Ong. Models of Higher-Order Computation: Recursion Schemes and Collapsible Pushdown Automata. In J. Esparza, B. Spanfelner, and O. Grumberg, editors, *Logics and Languages for Reliability and Security*, pages 263–300. IOS Press, 2010. NATO Science for Peace and Security Series, D: Information and Communication Security - Vol. 25.

19 C.-H. Luke Ong and Steven James Ramsay. Verifying Higher-Order Functional Programs with Pattern-Matching Algebraic Data Types. In *Proceedings of the Symposium on Principles of Programming Languages (POPL)*, pages 587–598. ACM, 2011.

20 John C. Reynolds. Definitional interpreters for higher-order programming languages. *Higher-Order and Symbolic Computation*, 11(4):363–397, 1998.

21 Hiroshi Unno, Naoshi Tabuchi, and Naoki Kobayashi. Verification of tree-processing programs via higher-order model checking. In *Proceedings of the Asian Symposium on Programming Languages and Systems (APLAS)*, volume 6461 of *Lecture Notes in Computer Science*, pages 312–327. Springer, 2010.