Turchin's Relation and Subsequence Relation in Loop Approximation

Antonina Nepeivoda

Program Systems Institute, Pereslavl-Zalessky, Russia
a_nevod@mail.ru

Abstract

The paper studies the subsequence relation through a notion of an intransitive binary relation on words in traces generated by prefix-rewriting systems. The relation was introduced in 1988 by V.F. Turchin for loop approximation in supercompilation. We study properties of this relation and introduce some refinements of the subsequence relation that inherit the useful features of Turchin's relation.

1 Introduction

In computer science the homeomorphic embedding is investigated from two completely different points of view, for it is of both theoretical and practical interest.

On the one hand, the embedding showed itself to be useful as a branch termination criterion in constructing tools for program transformation ([13], [2]). What makes the homeomorphic embedding reasonable as a termination criterion is the non-existence of an infinite sequence of finite labeled trees such that no tree in the sequence is embedded into some its derivative (the fact was proved by Kruskal and is called Kruskal's theorem; for an elegant proof of the fact see [7]). Relations with this property are called well-binary relations.

Definition 1. *R,$R \subset S \times S$, is called **a well binary relation**, if every sequence $\{\Phi_n\}$ of elements from S such that $\forall i,j(i < j \Rightarrow (\Phi_i, \Phi_j) \notin R)$ is finite. If R is well binary and transitive it is called a well quasiorder (wqo).*

*A sequence $\{\Phi_n\}$ with the property $\forall i,j(i < j \Rightarrow (\Phi_i, \Phi_j) \notin R)$ is called **a bad sequence** with respect to R. Thus, well-binariness of R can be formulated equivalently as "all bad sequences with respect to R are finite".*

On the other hand, well-binariness of the homeomorphic embedding is shown to be non-provable in the Peano arithmetic with the first-order induction scheme [12], and this fact aroused interest of logicians and computer scientists with background in mathematical logic (a thorough study of the proof-theoretical strength of the fact is in [16]). Studies of the homeomorphic embedding as a termination criterion for term rewriting systems ([11, 14]) are located in the middle between these poles of pure theory and practice.

The problem is that since these two domains live their own separate lives, it is not always obvious how to use the theoretical investigations in the practical program transformations. Theoricists study properties of the homeomorphic embedding (and similar relations) on arbitrary (maybe not even computable) sequences of trees, and that can imply somewhat obscure view on practical features of the relations: in particular it was established that the upper bound on a bad sequence length with respect to the homeomorphic embedding dominates every multiple recursive function [12], which looks redundantly from the practical point of view. But in real applications the opposite problem becomes much more frequent: the homeomorphic embedding yields branch termination too early [6, 11]. In some algorithms of program analysis this flaw was

partially fixed either by making an additional annotation [5] or by intersecting the embedding with other wqos [1].

In this paper we study properties of a special case of the homeomorphic embedding on a restricted set of computable sequences.

Definition 2. *Having two words Φ, Ψ in an alphabet Υ let us say that Φ is embedded in Ψ* ***with respect to the subsequence relation*** *($\Phi \trianglelefteq \Psi$) ($\trianglelefteq$ is also called the scattered subword relation) if Φ is a subsequence of Ψ.*

The subsequence relation is proved to be a well quasiorder by G. Higman [3]. We prove that while applied only to sequences generated by prefix grammars the relation admits bad sequences not more than exponential over a grammar size. If we apply the relation to a direct product of sequences generated by prefix grammars we receive the multiple recursive upper bound found by H. Touzet [14]. Also we show how to make a refinement of the subsequence relation that solves the empty word problem for languages generated by alphabetic prefix grammars and inherits some useful features of Turchin's relation, which was also used in program transformation (in particular, in the supercompiler SCP4 [8]).

The paper is organized as follows. First, we introduce notion of a prefix grammar. Then we give a definition of the Turchin relation and shortly prove its well-binariness. After that we show how to build maximal bad sequences with respect to the Turchin relation and give some discussion on using this relation combined with other well binary relations. Finally, we show how our refinement for the Turchin relation allows to refine the subsequence relation and, using our knowledge about the Turchin relation, we investigate properties of the subsequence relation on traces generated by prefix grammars.

The main contributions of the paper are the following:

1. We outline the concept of Turchin's relation in terms of prefix grammars and investigate properties of the relation.
2. We link Turchin's relation with the subsequence relation and show how to model the former by the latter not using a notion of time for sequences generated by prefix grammars.
3. We determine upper bounds of bad sequence length with respect to both relations for sequences generated by a single prefix grammar and for direct products of two sequences generated by prefix grammars.
4. We show that a minimal natural well binary generalization of Turchin's relation on direct products of sequences generated by prefix grammar is the subsequence relation.

2 Prefix Grammars

We consider a restricted class of generative indeterministic grammars, in which rewriting rules are applied in an arbitrary order.

Definition 3. *A tuple $\langle \Upsilon, \mathbf{R}, \Gamma_0 \rangle$, where Υ is an alphabet, $\Gamma_0 \in \Upsilon^+$ is an initial word, and $\mathbf{R} \subset \Upsilon^+ \times \Upsilon^*$ is a finite set of rewrite rules*[1]*, is called* ***a prefix grammar*** *if $R : R_l \to R_r$ can be applied only to words of the form $R_l\Phi$ (where R_l is a prefix and Φ is a (possibly empty) suffix) and generates only words of the form $R_r\Phi$.*

If the left-hand side R_l of a rule $R : R_l \to R_r$ has the length 1 (only the first letter is rewritten) then the prefix grammar is called ***an alphabetic prefix grammar.***

[1] It is usually said that Υ is finite, but this restriction is unnecessary in our case. Only finiteness of $\mathbf{R}$ matters in our study.

A trace *of a prefix grammar* $\mathbf{G} = \langle \Upsilon, \mathbf{R}, \Gamma_0 \rangle$ *is a word sequence* $\{\Phi_i\}$ *(finite or infinite) where* $\Phi_1 = \Gamma_0$ *and for all* i $\exists R(R : R_l \to R_r \ \& \ R \in \mathbf{R} \ \& \ \Phi_i = R_l\Theta \ \& \ \Phi_{i+1} = R_r\Theta)$ *(*Θ *is a suffix). In other words, the elements of a trace are derived from their predecessors by applications of rewrite rules from* $\mathbf{G}$.

Example 1. *Consider the following prefix grammar* $\mathbf{G_\Lambda}$ *with* $\Upsilon = \{a, b, c\}$ *and the following rewrite rules:*

$$R^{[1]} : \Lambda \to ba \qquad R^{[2]} : b \to \Lambda \qquad R^{[3]} : aac \to \Lambda$$
$$R^{[4]} : aad \to \Lambda$$

We cannot apply the rule $R^{[3]}$ *to* **baacb**, *for* **baacb** *starts not by* **aac**. *If we apply* $R^{[1]}$ *or* $R^{[2]}$ *to* **baacb** *the only correct results of the applications are* **babaacb** *and* **aacb** *respectively.*

When V. F. Turchin discussed a search of semantic loops in Refal programs he considered a stack model, which resembles a prefix grammar [15]. V. F. Turchin proposed to observe call stack configurations to prevent infinite unfolding of a special sort. He aimed at cutting off branches where a stack top derives a path that ends with the same stack top. If we denote the stack top as Φ, the derivation of Φ with Φ on the top as $\Phi\Psi$, and the part of the initial stack that is not modified as Θ then we can say that a branch is dangerous with respect to Turchin's relation if it contains pairs of the form $\Phi\Theta$, $\Phi\Psi\Theta$. We can notice that the terms form a pair with respect to the subsequence relation, but V. F. Turchin proposed a stronger relation for more precise identification of such stack configurations in his work [15]. V. F. Turchin used this relation to construct better loop approximations in residual programs, but the relation can be also used to forbid a program transformation process to halt driving on finite computation branches. The last property is analyzed in this paper for prefix-grammar-generated traces.

3 Turchin's Relation

To describe the Turchin relation for grammar-generated traces we use a formalization presented in [8]. The formalization introduces a notion of time indices. The main idea of the formalization is to mark every letter in the trace by a natural number that points to the position in the trace where the letter first appears. The order of words in a trace is from up to down.

The length of Φ is denoted by $|\Phi|$.

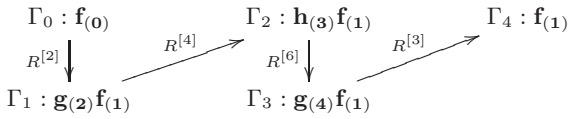
Definition 4. *Consider a trace* $\{\Phi_i\}$ *generated by a prefix grammar* $\mathbf{G}$, $\mathbf{G} = \langle \Upsilon, \mathbf{R}, \Gamma_0 \rangle$. *Supply letters of* Φ_i *by numbers that correspond to their* ***time indices*** *as follows. The* i*-th letter of* Γ_0 *is marked by the number* $|\Gamma_0| - i$*; if the maximal time index in the trace* $\{\Phi_i\}_{i=1}^{k}$ *is* M *and* Φ_{k+1} *is derived from* Φ_k *by an application of* $R : R_l \to R_r$ *then the* i*-th letter* Φ_{k+1} *(*$i \leq |R_r|$*) is marked by* $M + |R_r| - i + 1$. *Time indices of the other letters of* Φ_{k+1} *coincide with the corresponding time indices of* Φ_k.

We call such annotation time indexing and we call a trace with the annotation ***a computation***.

Example 2. *Let us consider a grammar* $\mathbf{G_{LOG}}$ *with* $\Upsilon = \{f, g, h\}$ *and the following rewrite rules:*

$$R^{[1]} : f \to \Lambda \qquad R^{[3]} : g \to \Lambda \qquad R^{[5]} : h \to \Lambda$$
$$R^{[2]} : f \to gf \qquad R^{[4]} : g \to h \qquad R^{[6]} : h \to g$$

$\Gamma_0 = f$. *A first segment of a computation yielded by the grammar* $\mathbf{G_{LOG}}$ *can look as:*



The time indices are in subscripts, enclosed in brackets. Note that the letter $f_{(0)}$ *in* Γ_0 *is replaced by* $f_{(1)}$ *in* Γ_1*, and* $f_{(0)} \neq f_{(1)}$.

In the sequel Greek capitals (Γ, Δ, Θ, Ψ, Φ) denote words in a computation (with the time indexing). $\Delta[k]$ denotes the k-th letter of Δ (counting from the beginning).

An equivalence up to the time indices is formally defined as follows. $\Phi \approx \Psi$ if $|\Phi| = |\Psi|$ and $\forall i(i \geq 1 \ \& \ i \leq |\Phi| \Rightarrow (\Phi[i] = a_{(n)} \ \& \ \Psi[i] = b_{(m)} \Rightarrow a = b))$. The definition has the following simple meaning: if we erase time indices of all letters in Φ and Ψ then Φ and Ψ will coincide literally. For instance, in Example 2 $f_{(0)} \neq f_{(1)}$, but $f_{(0)} \approx f_{(1)}$.

Now we are ready to define ***Turchin's relation*** $\Gamma \preceq \Delta$. Loosely speaking, it includes pairs $\langle \Gamma, \Delta \rangle$, where Γ can be presented as `[Top][Context]`, Δ can be presented as `[Top][Middle][Context]`, and the suffix `[Context]` is not modified in the computation segment thar starts from Γ and ends with Δ.

Definition 5. $\Gamma \preceq \Delta \Leftrightarrow \Gamma = \Phi\Theta_0 \ \& \ \Delta = \Phi'\Psi\Theta_0 \ \& \ \Phi' \approx \Phi$. *Pairs* Γ, Δ *such that* $\Gamma \preceq \Delta$ *are called* ***Turchin pairs***[2].

$\preceq$ is not transitive but it is reflexive and antisymmetric up to $\approx$ [10]. Well-binariness of the relation can be proved using the following observation. If a rule R has a non-empty right-hand side R_r, $\Phi \approx R_r$, $\Phi' \approx R_r$, $\Phi\Theta_0$ precedes $\Phi'\Theta_1$, and $\exists i(\Theta_1[i] = \Theta_0[1])$ then $\Phi\Theta_0 \preceq \Phi'\Theta_1$. So the maximal word length in a bad sequence with respect to $\preceq$ is bounded by

$$|\Gamma_0| + \sum(|R_r^{[i]}| - 1)$$

where Γ_0 is the initial word and $\sum(|R_r^{[i]}| - 1)$ runs over the set of different right-hand sides of all rules.

The upper bound is not exact due to the following two limitations. First, not every letter can be rewritten to the chosen right-hand side, i.e. the letter f cannot be rewritten to h in a one step. Second, some rules can accidentally share some letters in their right-hand sides. I.e. the letter g in the right-hand side of the rule $f \to gf$ and the letter g in the right-hand side of the rule $h \to g$ have different nature and the coincidence of the two letters is occasional. In the next section we show how to partly avoid this difficulty.

4 Annotated Prefix Grammars

If in the rules $h \to g$ and $f \to gf$ we write down the corresponding letters as e.g. $g^{[f]}$ and $g^{[h]}$ and say that $g^{[f]} \not\approx g^{[h]}$ then the prefix grammar will generate computations with less number of occasional Turchin's pairs.

Let us give more formal definition of this sort of prefix grammars.

[2] In [8] it is also specified that $|\Phi| > 0$. If a computation is yielded by a grammar only with non-empty left-hand sides of rules then this limitation is unnecessary. Otherwise the condition $|\Phi| > 0$ becomes essential to make the upper bound C'_{Max} constructed with a help of Lemma 1 exact.

Definition 6. *A prefix grammar* $\mathbf{G}=\langle \Upsilon, \mathbf{R}, \Gamma_0\rangle$, $\mathbf{R} \subset \Upsilon^+ \times \Upsilon^*$ *is called* ***annotated***[3] *if*

1. *For every two rules* $R : R_l \to R_r$, $R' : R'_l \to R'_r$, *if* $\exists i, j(R_r[i] \approx R'_r[j])$, *then* $R_r \approx R'_r$;
2. *If* $R_l \to R_r \in \mathbf{R}$ *and there is a rule* $R'_l \to R'_r$ *in* $\mathbf{R}$ *then* $R'_l \to R_r \in \mathbf{R}$.
3. *The initial word contains only unique letters:* $\forall i, j, k(R_r^{[k]}[i] \neq \Gamma_0[j])$.

Consider the following algorithm that transforms a prefix grammar $\mathbf{G}$ to an annotated $\mathbf{G}'$.

1. Let $a = R_r^{[n]}[i]$, $a \in \Upsilon$, n be an unique number of the rule with the right-hand side $R_r^{[n]}$. a corresponds to the pair $\langle a, 2^n * 3^{i-1}\rangle$. We set $n = 0$ for the initial word Γ_0 and denote the corresponding tuple of the pairs $\langle\Gamma_0[1], 1\rangle\langle\Gamma_0[2], 3\rangle \ldots \langle\Gamma_0[|\Gamma_0|], 3^{|\Gamma_0|}\rangle$ as Γ'_0.
2. A rewrite rule $R' : R'_l \to \Phi$ of the grammar $\mathbf{G}'$ corresponds the the equivalence class up to left-hand sides of rules $\langle a_i, n_i\rangle \to \Phi$, where Φ is a right-hand side of a rule from $\mathbf{G}$ after the first step, and $\langle a_i, n_i\rangle$ is arbitrary.

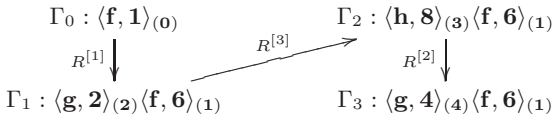
If the initial grammar $\mathbf{G}$ yields a bad sequence then the computation by $\mathbf{G}'$ that is derived from Γ'_0 by application of the rules from the equivalence classes that correspond to the right-hand sides of the rules that are applied in the computation by $\mathbf{G}$ is also a bad sequence.

We do not differ rewrite rules with the different left-hand sides in annotated grammars and write them as $x \to R_r$ where x denotes an arbitrary pair sequence of a bounded length.

Example 3. *Let us transform the prefix grammar* $\mathbf{G_{LOG}}$ *from Example 2 into an annotated.*

$\mathbf{G'_{LOG}}$*:*
$\Gamma_0 = \langle f, 1\rangle$
$R^{[1]} : x \to \langle g, 2\rangle\langle f, 6\rangle$
$R^{[2]} : x \to \langle g, 4\rangle$
$R^{[3]} : x \to \langle h, 8\rangle$
$R^{[4]} : x \to \Lambda$

The computation by $\mathbf{G_{LOG}}$ *that corresponds to the computation from Example 2 now begins as follows:*



Note that now $\Gamma_1 \not\preceq \Gamma_3$.

A useful feature of annotated grammars is their ability to generate longest bad sequences. There are no intersections in the right-hand sides of rewrite rules and thus $\approx$ discerns prefixes that are yielded by distinct rule applications. [4]

Now we can find the upper bound of a bad sequence length in a computation yielded by a prefix grammar. The proof uses the following lemma.

Lemma 1. *Every computation by an annotated prefix grammar ends either by* Λ *or by a Turchin pair* $\Phi\Theta_0$, $\Phi'\Psi\Theta_0$ *such that there exists a rule* $R_l \to R_r$, *for which* $\Phi \approx R_r$, $\Phi' \approx R_r$, *and* $R_r \neq \Lambda$.

[3]This grammar property 2 plays a role only in the construction of a longest bad sequences. So in most propositions grammars with only the properties 1 and 3 are also considered as annotated.

[4]This can be very useful if there is a rule R with the left-hand side R_l embedded in the right-hand side R_r. Note that if $R_l \trianglelefteq R_r$ then the subsequence termination criterion is always activated after an application of the rule $R_l \to R_r$. This problem was pointed in [11].

Proof. Let us consider a pair $\Phi_1\Theta_0$, $\Phi_2\Psi\Theta_0$ such that $\Phi_1\Theta_0 \preceq \Phi_2\Psi\Theta_0$, $(\Phi_1 \approx \Phi_2)$, and the trace segment ending with $\Phi_2\Psi\Theta_0$ is a bad sequence. According to the properties of annotated grammars, $\Phi_1[1]$ and $\Phi_2[1]$ must be generated by different applications of the same rule $R : x \to R_r$ with $|R_r| > 0$, and if $\Phi_1[1] \approx R_r[i]$ then necessarily $\Phi_2[1] \approx R_r[i]$. Let us denote the prefix $R_r[1]R_r[2]...R_r[i-1]$ as $R^{(i-1)}_{(z)}$ (z is the time index of $R_r[i-1]$). Now turn back to the two applications of R. The result of the former must be of the form $R^{(i-1)}_{(k_1)}\Phi_1\Theta_0$, the result of the latter is of the form $R^{(i-1)}_{(k_2)}\Phi_2\Psi\Theta_0$. They form a Turchin pair and therefore coincide with $\Phi_1\Theta_0$ and $\Phi_2\Psi\Theta_0$.

So $\Phi_1 = R_{r_1}\Phi'_1$, $\Phi_2 = R_{r_2}\Phi'_2$ ($\Phi'_1 \approx \Phi'_2$, and R_{r_1} and R_{r_2} coincide up to the time indices with some right-hand side of a rewrite rule). Let Φ'_1 be non-empty. Then $\exists R', j(\Phi'_1[1] \approx R'_r[j]$ & $\Phi'_2[1] \approx R'_r[j])$, and $\Phi'_1[1] \neq \Phi'_2[1]$. The prefix $R'_r[1]R'_r[2]...R'_r[j-1]$ is denoted as $R'^{(j-1)}_{(z)}$. Now turn back to the R' applications that generate $\Phi'_1[1]$ and $\Phi'_2[1]$. They look as $R'^{(j-1)}_{(l_1)}\Phi'_1\Theta_0$ and $R'^{(j-1)}_{(l_2)}\Phi'_2\Psi\Theta_0$ and form a Turchin pair. This contradicts the choice of $\Phi_1\Theta_0$ and $\Phi_2\Psi\Theta_0$.

Hence $\Phi_1\Theta_0 = R_{r_1}\Theta_0$ and $\Phi_2\Psi\Theta_0 = R_{r_1}\Psi\Theta_0$.

□

Note that the proof is for not only alphabetic prefix grammars but for prefix grammars that allow rules of the form $\Phi \to \Psi$. With the help of Lemma 1 we proved that the exact upper bound of a bad sequence length for an annotated prefix grammar is

$$C'_{Max} = |\Gamma_0| * (1 + |R^{[0]}_r| * (1 + |R^{[1]}_r| * (\cdots * (1 + |R^{[N]}_r|) \dots)))$$

where rules in the sequence $R^{[0]}, R^{[1]}, \dots, R^{[N]}$ are placed by a non-increasing order with respect to the length of their right-hand sides $|R^{[i]}_r|$ (the proof of this fact is by induction; for details see [10]).

Note that N in the formula $|\Gamma_0| * (1 + |R^{[0]}_r| * (1 + |R^{[1]}_r| * (\cdots * (1 + |R^{[N]}_r|) \dots)))$ denotes not the cardinality of the set of rewrite rules but the cardinality of the set of the right-hand sides of rewrite rules. Thus when we do the annotation there is no exponential growth of the upper bound.

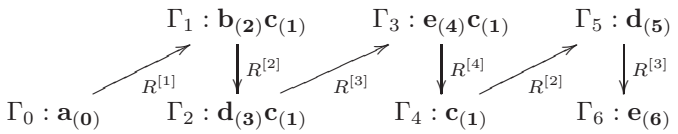
Example 4. *Let us estimate the length of a longest bad sequence yielded by the grammar* $\mathbf{G'_{LOG}}$ *(Example 3). The length of the initial word is* 1, $|R^{[1]}_r|$ *has the length* 2, *and two rules have the right-hand sides of the length* 1. *The corresponding bad sequence length is* 7.

Now let us build such bad sequence explicitly. For the sake of readability different pairs of the form ⟨ *letter, number*⟩ *are denoted by different letters (thus* $\langle f, 1\rangle = a$, $\langle g, 2\rangle = c$, $\langle f, 6\rangle = c$, $\langle g, 4\rangle = d$, *and* $\langle h, 8\rangle = e$ *).*

$\mathbf{G'_{LOG}}$*:*

$\Gamma_0 = a$ $\quad R^{[1]} : x \to bc$ $\quad R^{[2]} : x \to d$ $\quad R^{[3]} : x \to e$ $\quad R^{[4]} : x \to \Lambda$

One of the maximal bad sequences is:



Note that the segment Γ_5–Γ_6 *cannot be generated by the initial grammar* $\mathbf{G_{LOG}}$.

If we aim to find embeddings not only in traces generated by single prefix grammars but also in direct products of the traces then usage of $\preceq$ causes some questions. Namely we must know whether well-binariness is preserved on intersections of the Turchin relation with some wqo. The problem is that the Turchin relation is not well binary on arbitrary computations' subsequences — we only can prove that it is well binary on the whole computations.

Example 5. *Consider the following computation yielded by a prefix grammar.*

Γ_0 :	$\mathbf{a_{(2)}b_{(1)}c_{(0)}}$	Γ_7 :	$\mathbf{b_{(5)}c_{(3)}}$
Γ_1 :	$\mathbf{b_{(1)}c_{(0)}}$	Γ_8 :	$\mathbf{c_{(3)}}$
Γ_2 :	$\mathbf{c_{(0)}}$	Γ_9 :	$\mathbf{b_{(10)}c_{(9)}}$
Γ_3 :	$\mathbf{b_{(4)}c_{(3)}}$	Γ_{10} :	$\mathbf{a_{(12)}b_{(11)}c_{(9)}}$
Γ_4 :	$\mathbf{a_{(6)}b_{(5)}c_{(3)}}$	Γ_{11} :	$\mathbf{a_{(14)}a_{(13)}b_{(11)}c_{(9)}}$
Γ_5 :	$\mathbf{a_{(8)}a_{(7)}b_{(5)}c_{(3)}}$	Γ_{12} :	$\mathbf{a_{(16)}a_{(15)}a_{(13)}b_{(11)}c_{(9)}}$
Γ_6 :	$\mathbf{a_{(7)}b_{(5)}c_{(3)}}$	...	...

No two elements of the sequence Γ_0, Γ_5, Γ_{12}, Γ_{21}, ... *form a Turchin pair.*

The following lemma verifies well-binariness of the intersections.

Lemma 2. $\preceq$ *contains a wqo* T *that is well binary on all computations yielded by an annotated prefix grammar.*

Proof. Let $\langle \Upsilon, \mathbf{R}, \Gamma_0 \rangle$ be an annotated prefix grammar $\mathbf{G}$. Consider all traces $\{\Phi_i\}_{i=1}^{\infty}$ generated by $\mathbf{G}$ such that $\exists N \, \forall i \, \exists j (i < j \ \& \ |\Phi_j| \leq N)$.

For every trace J from this set choose the least N that satisfies this property. Due to finiteness of the set $\mathbf{R}$ words generated by the rules from $\mathbf{R}$ can contain finite set of letters. Therefore some word Ψ of the length N must repeat itself (with respect to $\approx$) infinitely in J. The first letter of Ψ is generated by a single rule R with a non-empty right-hand side. Every two results of these applications of R look as $\Delta\Psi$ and $\Delta'\Psi'$ where $\Psi \approx \Psi'$ and $\Delta \approx \Delta'$ for they are same prefixes of the same right-hand side R_r that end at $\Psi[1]$ so $\Delta\Psi$ and $\Delta'\Psi'$ form a Turchin pair.

All other traces $\{\Phi_i\}_{i=1}^{\infty}$ have an infinite growth of the minimal word length: $\forall N \, \exists i_N \, \forall j (j > i_N \Rightarrow |\Phi_j| > N)$. For every such trace and every N choose a minimal i_N such that all successors of Φ_{i_N} never have the length less than N: $\forall j (j < i_N \Rightarrow \exists k (k \geq j \ \& \ |\Phi_k| < N))$. So $|\Phi_{i_N - 1}| < N$, $|\Phi_{i_N}| \geq N$, and Φ_{i_N} is generated from its predecessor by some R with a non-empty right-hand side, $|R_r| \geq 2$: $\Phi_{i_N} = R_{r(l)}\Phi^{-}_{i_N - 1}$ where $\Phi^{-}_{i_N - 1}$ is a suffix of $\Phi_{i_N - 1}$. $\Phi^{-}_{i_N - 1}$ stays constant because $|\Phi_{i_N - 1}| < N$. All the elements of $\{\Phi_{i_N}\}_{N=1}^{\infty}$ begin with a non-empty right-hand side of a some rewrite rule, therefore exists an infinite subsequence $\{\Phi_{i_K}\}_{K=1}^{\infty}$ of $\{\Phi_{i_N}\}_{N=1}^{\infty}$ such that all elements of $\{\Phi_{i_K}\}_{K=1}^{\infty}$ begin with the right-hand side of a same rule. Every two elements of $\{\Phi_{i_K}\}_{K=1}^{\infty}$ form a Turchin pair.

The set T of pairs of these two sorts is a wqo on traces generated by an annotated prefix grammar. □

T contains Turchin pairs of a special sort. Namely it contains $\langle \Gamma, \Delta \rangle$ such that $\Gamma = \Phi\Theta_0$, $\Delta = \Phi'\Psi\Theta_0$, there exists a rule $R : R_l \to R_r$ with $|R_r| > 0$, $R_r \approx \Phi$, $R_r \approx \Phi'$, and $\Psi[1]$ is never modified in the further computation. So the Turchin relation may not be checked after erasures with no loss of well-binariness.

What is more, existence of T guarantees that the Turchin relation can be intersected with an arbitrary wqo without loss of well-binariness. On the other hand, the idea of intersecting two Turchin relations looks appealing but implies a possible existence of infinite bad sequences with respect to the intersection.

Definition 7. *Let us say that* Γ *is embedded in* Δ *with not more than with a single gap if there exist words* Φ, Ψ, Θ *(maybe empty) such that* $\Gamma = \Phi\Theta$, $\Delta = \Phi\Psi\Theta$.

Let us say that Γ *is embedded in* Δ ***with not more than with*** $n+1$ ***gaps*** *if there exist (maybe empty)* Φ, Ψ, Θ_1, Θ_2 *such that* $\Gamma = \Phi\Theta_1$, $\Delta = \Phi\Psi\Theta_2$ *and* Θ_1 *is embedded in* Θ_2 *with not more than with* n *gaps.*

Let us give a simple example. **abac** is embedded in **abrac** with not more than a single gap and in **abracadabra** — with not more than two gaps (**abac** is divided into only two parts **ab** and **ac**, but the end of a word is also considered as its part. The end of the word **abracadabra** is not at the same position as the end of **ac**, so the gap between **ac** and the end of the word is also taken into account).

Lemma 3. *If a relation* R *of word embedding allows only finite number of gaps then it is not well binary on sequences that are yielded by a direct product of prefix grammars* $G_1 \times G_2$*, even when the grammars are deterministic.*

Proof. Let us consider a class of grammars $\mathbf{G}^{[n]}$ on pairs of words in the alphabet $\{a_1, \ldots, a_n, A_1, \ldots, A_n, e, E\} \times \{a_1, \ldots a_n, A_1, \ldots, A_n, e, E\}$.

Let the initial word be $\langle e, A_1 A_2 \ldots A_n E\rangle$ and $\mathbf{R}$ consist of the following rewrite rules:

$R^{[00]} : \langle e, a_1\rangle \to \langle E, a_1 a_1\rangle$
$R^{[01]} : \langle E, A_1\rangle \to \langle e, A_1 A_1\rangle$
$R^{[02]} : \langle e, A_1\rangle \to \langle a_1 e, \Lambda\rangle$
$R^{[03]} : \langle E, a_1\rangle \to \langle A_1 E, \Lambda\rangle$
...
$R^{[i0]} : \langle a_i, a_i\rangle \to \langle \Lambda, a_i a_i\rangle$
$R^{[i1]} : \langle A_i, A_i\rangle \to \langle \Lambda, A_i A_i\rangle$
$R^{[i2]} : \langle a_i, A_i\rangle \to \langle a_i a_i, \Lambda\rangle$
$R^{[i3]} : \langle A_i, a_i\rangle \to \langle A_i A_i, \Lambda\rangle$
$R^{[i4]} : \langle a_i, a_{i+1}\rangle \to \langle \Lambda, a_i a_{i+1} a_{i+1}\rangle$
$R^{[i5]} : \langle A_i, A_{i+1}\rangle \to \langle \Lambda, A_i A_{i+1} A_{i+1}\rangle$
$R^{[i6]} : \langle a_i, A_{i+1}\rangle \to \langle a_{i+1} a_i a_i, \Lambda\rangle$
$R^{[i7]} : \langle A_i, a_{i+1}\rangle \to \langle A_{i+1} A_i A_i, \Lambda\rangle$
...
$R^{[n0]} : \langle a_n, e\rangle \to \langle \Lambda, a_n e\rangle$
$R^{[n1]} : \langle A_n, E\rangle \to \langle \Lambda, A_n E\rangle$
$R^{[n2]} : \langle a_n, E\rangle \to \langle a_n a_n, e\rangle$
$R^{[n3]} : \langle A_n, e\rangle \to \langle A_n A_n, E\rangle$

For every N there exists some n such that $\mathbf{G}^{[n]}$ yields a trace with no pair $\langle \Phi_1, \Psi_1\rangle$, $\langle \Phi_2, \Psi_2\rangle$ such that Φ_1 is embedded in Φ_2 and Ψ_1 is embedded in Ψ_2 with not more than N gaps. □

So the Turchin relation can be intersected with any relation that is well binary on the whole $\{\Upsilon^*\}$, but not with the other Turchin relation.

5 Turchin's Relation and Subsequence Relation

The Turchin theorem not only guarantees existence of a Turchin pair for every infinite computation but also gives the exponential upper bound of a bad sequence length. In the case of computations yielded by annotated prefix grammars the upper bound of a bad sequence with respect to the subsequence relation coincides with the upper bound of a bad sequence

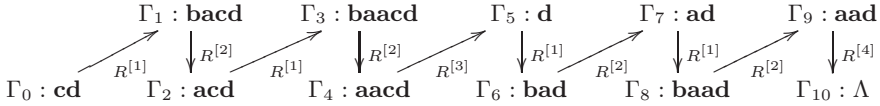
$C'_{Max} = |\Gamma_0| * (1 + |R_r^{[0]}| * (1 + |R_r^{[1]}| * (\cdots * (1 + |R_r^{[N]}|) \dots)))$ for the Turchin relation. What is more, in a computation yielded by an annotated prefix grammar the lengths of bad sequences with respect to these two relations always coincide.

Lemma 4. *The first pair in a computation yielded by an annotated prefix grammar that satisfy the subsequence relation is a Turchin pair.*

Proof. Consider Φ_1 and Φ_2 such that $\Phi_1 \trianglelefteq \Phi_2$, there are no pairs with respect to the subsequence relation in the trace before Φ_2, and Φ_1 is embedded into Φ_2 with $n+1$ gaps (up to time indices). Let Θ_0 be their common suffix. Then $\Phi_1 = A_1 A_2 \dots A_n \Theta_0$ and $\Phi_2 = B_1 A'_1 B_2 A'_2 \dots B_n A'_n B_{n+1} \Theta_0$, where $A_i \approx A'_i$ for all i from 1 to n. Consider the words where letters $A_n[1]$ and $A'_n[1]$ are generated. The grammar features guarantee that both letters are generated by a same rule $R : x \to R_r$, so the words look as $\Delta A_n \Theta_0$ and $\Delta' A'_n B_{n+1} \Theta_0$. $\Delta \approx \Delta'$ for they are same prefixes of the same right-hand side. This implies that $\Delta A_n \Theta_0 \trianglelefteq \Delta' A'_n B_{n+1} \Theta_0$. In the computation Φ_1 and Φ_2 are the first pair with respect to the subsequence relation, consequently $i = 1 = n$, and $\Phi_1 = A_1 \Theta_0$, and $\Phi_2 = A'_1 B_{n+1} \Theta_0$, so $\Phi_1 \trianglelefteq \Phi_2$. □

Lemma 4 may have some practical meaning for systems of program transformation that use the homeomorphic embedding as a branch termination criterion. We discuss this option in Section 6.

Then the question emerges if the annotated subsequence relation can prevent too early terminations for every prefix-grammar-generated computation. Namely, whether the annotated subsequence relation allows unfolding to find a trace ending by Λ if Λ is in the language of the prefix grammar. The answer is yes for alphabetic prefix grammars [9] and is negative in the general case when $\mathbf{R} \subset \Upsilon^* \times \Upsilon^*$. To illustrate the last claim, consider the grammar $\mathbf{G_\Lambda}$ of Example 1 if $\Gamma_0 = cd$. Λ belongs to the language of the grammar since



The grammar belongs to the class of annotated grammars. But there are several pairs with respect to $\trianglelefteq$ (and $\triangleleft$) on the trace leading to Λ: e. g. $\Gamma_1 \triangleleft \Gamma_3$. To solve the empty word problem for languages generated by non-alphabetic prefix grammars using the subsequence relation as a termination criterion we need to do some more annotation, which is proved in [9].

Recall that in the case of pairs over $\trianglelefteq$ the upper bound on arbitrary sequences with restricted word length growth is multiple recursive [14]. We show that the upper bound is exact even if the sequences are built by a direct product of two prefix grammars.

Example 6. *Consider the following rewrite grammar (it not necessarily rewrites only prefixes; the grammar is similar to the one described in [14]).*

$R^{[1]} : su \to ss$ $R^{[2]} : tu \to tt$ $R^{[3]} : ts \to tt$
$R^{[4]} : wu \to ws$ $R^{[5]} : tws \to utw$ $R^{[6]} : tw \to ws$
$R^{[7]} : sws \to wt$ $R^{[8]} : sw \to wsss$

If the rules are applied to the initial word $sss \dots swww \dots w$ then the trace of the length $O(B(m,n))$ with no pairs with respect to $triangleleftteq$ is generated.

Now we build a system of two prefix grammars that models the example of H. Touzet (x denotes an arbitrary letter).

1. *The first letter of a word rewritten by the first prefix grammar* $\mathbf{G_1}$ *represents a current state of the Turing machine.*
2. *The last letter of a word rewritten by the second prefix grammar* $\mathbf{G_2}$ *represents the end of data* $[EOW]$ *and is always rewritten into itself.*
3. *The word rewritten by* $\mathbf{G_1}$ *represents the initial fragment of data which is before the counter of Turing machine. The word rewritten by* $\mathbf{G_2}$ *represents the final fragment of data which is behind the counter of Turing machine.*
4. *There is an auxiliary set of rules moving the counter to the beginning of the data* $\langle State_0 x, y\rangle \to \langle State_0, xy\rangle$.
5. *There is a rule that starts the rewrite process* $\langle State_0, y\rangle \to \langle State_1^F, y\rangle$.
6. $R^{[i]} : R_l^{[i]} \to R_r^{[i]}$ *are modeled by* $\langle State_i^F x, [EOW]\rangle \to \langle State_i^B x, [EOW]\rangle$ *(if* $i \neq 8$*), a rule* $\langle State_i^F R_l^{[i]}, x\rangle \to \langle State_0\Lambda, R_r^{[i]} x\rangle$ *and a set of rewrite rules* $\langle State_i^F x, y\rangle \to \langle State_i^F xy, \Lambda\rangle$ *where* x *does not coincide with* $R_l^{[i]}$.
7. *The set of rules* $\langle State_i^B x, y\rangle \to \langle State_i^B, xy\rangle$ *is similar to the one for* $State_0$ *but the last rule now looks like* $\langle State_i^B, y\rangle \to \langle State_{i+1}^F, y\rangle$ *instead of* $\langle State_0, y\rangle \to \langle State_1^F, y\rangle$.

If there are two pairs $\langle a_1\Phi, \Psi\rangle$, $\langle a_2\Phi', \Psi'\rangle$ *such that* $a_1\Phi \trianglelefteq a_2\Phi'$ *and* $\Psi \trianglelefteq \Psi'$ *then* $a_1 = a_2$ *and* $\Phi\Psi \trianglelefteq \Phi'\Psi'$. *There can be no such pairs if* a_2 *is not changed on the trace fragment from* $\langle a_1\Phi, \Psi\rangle$ *to* $\langle a_2\Phi', \Psi'\rangle$ *because then* $|\Psi'| < |\Psi|$. *If* a_2 *is changed on the trace fragment from* $\langle a_1\Phi, \Psi\rangle$ *to* $\langle a_2\Phi', \Psi'\rangle$ *then one of the rules* $\langle State_i^F R_l^{[i]}, x\rangle \to \langle State_0\Lambda, R_r^{[i]} x\rangle$ *is applied on the fragment and thus* $\Phi\Psi \not\trianglelefteq \Phi'\Psi'$. *Therefore the bad sequence length on the trace generated by* $\mathbf{G_1} \times \mathbf{G_2}$ *with respect to the subsequence relation must be also estimated by* $O(B(m, n))$.

6 Possible Practical Applications

Let us see how the annotation can help to do more precise program analysis when the analyzed program is unfolded in the call-by-value style.

Example 7. *Consider a program that computes the least power of* 2 *that is greater than input. Let us build a prefix grammar that describes the call stack behavior when the program is executed in the call-by-value style.*

Function definitions		***Rewrite rules***
`f(Z)=S(Z);`	$\Rightarrow$	$f \to \Lambda$
`f(S(x))=S(f(g(S(Z))));`	$\Rightarrow$	$f \to gf$
`g(Z)=Z;`	$\Rightarrow$	$g \to \Lambda$
`g(S(x))=h(x);`	$\Rightarrow$	$g \to h$
`h(Z)=Z;`	$\Rightarrow$	$h \to \Lambda$
`h(S(x))=S(g(x));`	$\Rightarrow$	$h \to g$

If we want to unfold a semantic tree of the program runs on the call `f(S(S(x)))` *(with an indefinite parameter* `x`*) then* $\Gamma_0 = f$. *The initial segment of the trace looks as:*
$f \to gf \to hf \to gf$. *Then an ambiguity appears:* x *can have the value either* Z *or* $S(x')$, *so there are multiple possible traces representing the ways in the corresponding semantic tree.*

The order of letters in the rewrite rules of Example 7 is univocal. In the following situation:

```
f(Z)=Z;
f(S(x),y)=f(g(x),h(y));
```

an ambiguity appears. There is no explicit rule in what order the two operands of the outermost function `f` are placed in a stack so the corresponding rule can look either as $f \to ghf$ or $f \to hgf$.

In the case of the call-by-value execution style (i.e. as in Refal) we can straightforwardly build a grammar that describes a stack behavior. Namely we must do the following two actions:

1. For every function in the program determine a letter from Υ that represents the function name (arities of the functions are ignored).
2. For every function definition in the program determine a rewrite rule that represents it as follows. A letter that represents the name of the defined function is placed to the left-hand side of the rule and the word consisting of the corresponding letters in the order from the innermost to the outermost function call is placed to the right-hand side of the rule.

If there are no function calls in the left-hand sides of definitions then the algorithm generates an alphabetic prefix grammar. The grammar representation is consistent with a call stack behavior, though incomplete.

Now we can see with the help of Example 7 how annotating the subsequence relation can help to get a finite computation branch to be computed when the usual subsequence relation as a termination criterion makes the branch to construct a syntactic loop.

Example 8. *The definition line* `f(S(x))=S(f(g(S(x))))` *has a "badness" of the discussed sort: its left-hand side is embedded in the right-hand side in the sense of the subsequence relation. So even an unfolding of the call $f(S(Z))$ yields a pair with respect to $\trianglelefteq$ and if the subsequence relation plays a role of termination criterion then the unfolding will be terminated too early. If we use the Turchin relation on stack configurations then the computation is analyzed as follows (the second column represents the call stack configuration):*

`f(S(Z))`	$f_{(0)}$
`S(f(g(S(Z))))`	$g_{(2)}f_{(1)}$
`S(f(h(Z)))`	$h_{(3)}f_{(1)}$
`S(f(Z))`	$f_{(1)}$
`S(Z)`	Λ

Now let us try to unfold the computation of $f(S(S(Z)))$.

`f(S(S(Z)))`	$f_{(0)}$
`S(f(g(S(S(Z))))))`	$g_{(2)}f_{(1)}$
`S(f(h(S(Z))))`	$h_{(3)}f_{(1)}$
`S(f(S(g(Z))))`	$g_{(4)}f_{(1)}$
`S(f(S(Z)))`	$f_{(1)}$
`S(S(f(g(S(Z)))))`	$g_{(6)}f_{(5)}$
`S(S(f(h(Z))))`	$h_{(7)}f_{(5)}$
`S(S(f(Z)))`	$f_{(5)}$
`S(S(Z))`	Λ

Note that the annotated Turchin relation (not intersected with the subsequence relation on the whole term) terminates the computation on the pair $\langle S(f(g(S(S(Z))))))), S(S(f(g(S(Z))))))\rangle$.

But if we consider the intersection of these two relations as a termination criterion, not only the computation of $f(S(S(Z)))$ but a computation of every call $f(x)$ where x is a unary Peano number is a bad sequence with respect to the intersection [5]*. The other way to prevent too early termination is to use the annotated subsequence relation: it is only enough to annotate function calls to avoid unwanted embeddings with the effect similar to the usage of the Turchin relation intersected with $\trianglelefteq$.*

7 Conclusion

Now we can see that the transition from only stack transformations to stack-plus-data transformations even in the unary case lifts the computational model from the finite automata up to full power of Turing machines. Thus it becomes interesting to investigate how the popular wqos work on intermediate prefix grammar constructions (as -2 or -1-class prefix grammars [4]), which are widely used in term rewriting theory.

Acknowledgments

The author is grateful to A. P. Nemytykh for fruitful discussions and inspiration on investigating properties of the Turchin relation.

References

[1] E. Albert, J. Gallagher, M. Gomes-Zamalla, and G. Puebla. Type-based homeomorphic embedding for online termination. *Journal of Information Processing Letters*, 109(15):879–886, 2009.

[2] M. C. Bolingbroke, S. L. Peyton-Jones, and D. Vytiniotis. Termination combinators forever. In *Proceedings of the 4th ACM SIGPLAN Symposium on Haskell*, pages 23–34. Tokyo, 2011.

[3] G. Higman. Ordering by divisibility in abstract algebras. *Bulletin of London Mathematical Society*, 3(2):326–336, 1952.

[4] P. Jancar and J. Srba. *Undecidability Results for Bisimilarity on Prefix Rewrite Systems*, volume 3921 of *Lecture Notes in Computer Science*, pages 277–291. IEEE Computer Society Press, 2006.

[5] I. Klyuchnikov. Inferring and proving properties of functional programs by means of supercompilation. Ph. D. Thesis, 2010.

[6] M. Leuschel. *Homeomorphic Embedding for Online Termination of Symbolic Methods*, volume 2566 of *Lecture Notes in Computer Science*, pages 379–403. IEEE Computer Society Press, 2002.

[7] C. St. J. A. Nash-Williams. On well-uasi-ordering infinite trees. *Proceedings of Cambridge Philosophical Society*, 61:697–720, 1965.

[8] A. P. Nemytykh. *The Supercompiler Scp4: General Structure*. URSS, Moscow, 2007.

[9] A. Nepeivoda. Ping-pong protocols as prefix grammars and turchin's relation. In *VPT 2013. First International Workshop on Verification and Program Transformation*, volume 16, pages 74–87. EPiC Series, EasyChair, 2013.

[10] A. N. Nepeivoda. Turchin's relation and loop approximation in program analysis. In *Proceedings on the Functional Language Refal*, pages 170–192. Sbornik, Pereslavl–Zalessky, 2014.

[11] L. Puel. Using unavoidable set of trees to generalize kruskal's theorem. *Journal of Symbolic Computation*, 8:335–382, 1985.

[12] S. Simpson. Nonprovability of certain combinatorial properties of finite trees. *Harvey Friedmans research on the foundations of mathematics*, pages 87–117, 1985.

[5] A termination criterion of this type is used in the supercompiler SCP4 [8].

[13] M. H. Sørensen and R. Glück. An algorithm of generalization in positive supercompilation. In *Proceedings of ILPS'95, the International Logic Programming Symposium*, pages 465–479. MIT Press, 1995.

[14] H. Touzet. A characterisation of multiply recursive functions with higman's lemma. *Information and Computation*, 178:534–544, 2002.

[15] V.F. Turchin. The algorithm of generalization in the supercompiler. *Partial Evaluation and Mixed Computation*, pages 341–353, 1988.

[16] A. Weiermann. Phase transition thresholds for some friedman-style independence results. *Mathematical Logic Quarterly*, 53:4–18, 2007.