# On Unfolding for Programs Using Strings as a Data Type

Andrei P. Nemytykh[*]

Program Systems Institute of Russian Academy of Sciences
`nemytykh@math.botik.ru`

**Abstract**

As a rule, program transformation methods based on operational semantics unfold a semantic tree of a given program. Sometimes that allows ones to optimize the program or to automatically prove its certain properties. Unfolding is one of the basic operations, which is a meta-extension of one step of the abstract machine executing the program. This paper is interested in unfolding for programs based on pattern matching and manipulating the strings. The corresponding computation model originates with Markov's normal algorithms and extends this theoretical base. Even though algorithms unfolding programs were being intensively studied for a long time in the context of variety of programming languages, as far as we know, the associative concatenation was stood at the wayside of the stream. We define a class of term rewriting systems manipulating with strings and describe an algorithm unfolding the programs from the class. The programming language defined by this class is Turing-complete. The pattern language of the program class in turn fixes a class of word equations. Given an equation from the class, one of the algorithms suggested in this paper results in a description of the corresponding solution set.

**Keywords:** Program specialization, supercompilation, program analysis, program transformation, unfolding, driving, Markov's normal algorithms, word equations.

## 1 Introduction

Let $\mathcal{L}$ be a functional/logical programming language. Given a program `P` written in $\mathcal{L}$, as a rule, program transformation methods based on operational semantics unfold and try to analyze a semantic tree of `P` in a given context of using `P`. If a program transformation method $\mathcal{M}$ aims to optimize `P` w.r.t. its run-time, then an abstract cost model is required and the optimizer trying to reduce the run-time has to follow the model. Usually the $\mathcal{L}$-interpreter is implemented by means of a staircase of other languages. I.e. the $\mathcal{L}$-interpreter is written in a language $\mathcal{L}_1$ implemented in $\mathcal{L}_2$ and so on. This circumstance together with specific properties of the hardware executing the program `P` make rather impossible using an actual (physical) execution time-cost model in theoretical investigations.

The computation model standing behind the operational $\mathcal{L}$-semantics gives a natural *logical* time-cost model widely used in program transformation. The corresponding logical time-cost of $\mathtt{P(d_0)}$ is the number of computation steps taken by evaluation of $\mathtt{P(d_0)}$. Such a computation `Step` defined by the operational semantics is a subroutine being iterated by the $\mathcal{L}$-interpreter `Int` in its uppermost loop. Given a current state `st` of `Int`, the current call `Step(st)` results in the next state following immediately after `st`. In general, the actual execution time taken by the step evaluation is not uniformly bounded above by the size of the input data and this fact may be unobvious without knowledge of some details of the entire implementation staircase

[*]The reported study was partially supported by RFBR, research project No. 14-07-00133_A, and by Program No. 16 for Basic Research of Presidium of Russian Academy of Sciences.

mentioned above[1]. Nevertheless such a time-cost model is interesting from both theoretical and practical points of view.

`Int` iterates `Step(st`$_0$`)` when its input data `st`$_0$ is completely static (known), while the unfolding discussed above generates the semantics tree by means of iterating a meta-extension `MStep(st)` of `Step` in the case when its input data `st` may be parameterized (partially unknown/dynamic). A launch of `MStep` results in a tree being branched with the corresponding input parameters. This tree is finite in the case when it is assumed that evaluation of `Step(st)` takes a time uniformly bounded by the size of `st`, otherwise the tree may be potentially infinite. Prolog-III [3] including a string unification is an example of the language where the time taken by the call `MStep(st)` is not uniformly bounded above. Associativity of the string concatenation makes this property syntactically explicit [20]. Another example is a functional programming language Refal [20, 22] not widely known; the Refal concatenation constructor is also associative. Constraint logic programs manipulating the string values are used in industrial programming. Strings form a fundamental data type used in most programming languages. Recently, a number of techniques producing string constraints have been suggested for automatic testing [8, 1] and program verification [16]. String-constraint solvers are used in many testing and analysis tools [5, 2, 6, 15, 23].

All of that attracts an interest to string manipulation in the context of program analysis and transformation. Even though algorithms unfolding programs were being intensively studied for a long time (see for examples: [18, 19, 10, 14, 7]), as far as we know, the associative concatenation was stood by the wayside of the stream and mainly was considered only in the context of the Refal language mentioned above [18, 19, 21].

In this paper we are interested in unfolding for programs based on pattern matching and manipulating the strings. The corresponding computation model originates with Markov's normal algorithms [13] and extends this theoretical base. Given a string, the pattern matching looks for an instance of a particular substring within the given string, and the logical time-cost model, by definition, regards this search as a basic operation taking *the only unit* of the logical time. Unfolding for such programs connects the program transformation with solving word equations – another fundamental problem arisen by A. A. Markov. The corresponding satisfiability problem was solved by G. S. Makanin [12], but it still remains nontrivial from a practical point of view. Unfolding for programs using strings as a data type is intensively used in a particular approach to the program transformation and specialization, known as supercompilation[2]. Supercompilation is a powerful semantics-based program transformation technique [17, 19, 18, 11] having a long history well back to the 1960-70s, when it was proposed by V. Turchin.

There is a number of simple algorithms dealing with specific classes of the word equations (see [4] for examples). Given a word equation, they describe, in a certain way, the corresponding solution set. The main problem is to create a formal description language which is both clear in details and allowing us to represent some infinite solution sets in finite terms. For instance any word equation itself is a description of its solution set, but, as a rule, such a description is very obscure.

**Our contribution.** We give a syntactic description of a class of term rewriting systems manipulating with strings and using the theoretical time-cost model mentioned above. We suggest a new algorithm for one-step unfolding of the programs from this class and this algorithm handles an input that is partially unknown. I.e. we present a driving operation. The programming language defined by this class is Turing-complete. The class is studied both in nondeterministic and deterministic variants of the operational semantics. Given a word equation, one of the

[1]For example (but not only), a hidden garbage collection may cause that.

[2]From *super*vised *compilation*.

algorithms suggested in this paper results in a description (a term rewriting system) of the corresponding solution set. As far as we know, there exist no papers using the nondeterministic pattern matching as a language for such a description and none considered the class of the word equations characterized by Theorems 1, 2 proved below, namely the variables separated by the two[3] equation sides such that at least one of the sides has at most one occurrence of each word variable.

The paper is organized as follows. Sect. 2 provides the program presentation languages. In Sect. 3 we consider important examples aiming to clarify the reader intuition on the unfolding over the strings. The two theorems being our main result are proved in Sect. 4. The theorems relate to both specialization of the programs manipulating the strings and solving the word equations. The constructive proofs of these theorems yield outlines of the algorithms unfolding the programs and solving the word equations from the class of the programs/(word equations) meeting the conditions of the theorems and mentioned above.

## 2 Preliminaries

### 2.1 The Presentation Language

We present our program examples in two variants of a pseudocode for functional programs. The first one named $\mathcal{L}$ is a deterministic language, while the second denoted $\mathcal{L}_*$ is nondeterministic one. The programs given below are written as *strict* term rewriting systems based on pattern matching. The sentences in the $\mathcal{L}$-programs are ordered from the top to the bottom to be matched. The sentences in the $\mathcal{L}_*$-programs are not ordered. The data set is a free monoid under concatenation. I.e. the concatenation is associative. The double-plus sign stands for the concatenation. The constant $\lambda$ is the identity element of the concatenation and may be omitted, the other constants $c$ are characters. Let $\mathcal{C}$ denote a set of the characters. The monoid of the data may be defined with the following grammar: $d$ `::=` $\lambda$ `|` $c$ `|` $d_1$ `++` $d_2$
Thus a datum is a finite sequence (including the empty sequence $\lambda$).

Let $\mathcal{F} = \cup_i \mathcal{F}_i$ be a finite set of functional symbols, here $\mathcal{F}_i$ is a set of functional symbols of arity $i$. Let $v, f$ denote a variable and a function name correspondingly, then the monoid of the corresponding terms may be defined as follows:
$t$ `::=` $\lambda$ `|` $c$ `|` $v$ `|` $f$`(` $args$ `)` `|` $t_1$ `++` $t_2$
$args$ `::=` $t$ `|` $t$`,` $args$ – where the number of the arguments of `f` equals its arity.

Let the denumerable variable set $\mathcal{V}$ be disjoined in two sets $\mathcal{V} = \mathcal{E} \cup \mathcal{S}$, where the names from $\mathcal{E}$ are prefixed with 'e.', while the names from $\mathcal{S}$ – with 's.'. $s$.variables range over characters, while $e$.variables range over the whole data set.

We refer to unknown data as *parameters*. Like the variables the parameters $p$ may be one of the two following types – `e` and `s`: $p$ `::= #s.name | #e.name`. They range similarly to the corresponding variables. Unlike the variables the parameters are semantics entities.

For a term $t$ we denote the set of all e-variables (s-variables) in $t$ by $\mathcal{E}(t)$ (correspondingly $\mathcal{S}(t)$), the set of all e-parameters (s-parameters) in $t$ by ${}^{\#}\mathcal{E}(t)$ (correspondingly ${}^{\#}\mathcal{S}(t)$). $\mathcal{V}(t) = \mathcal{E}(t) \cup \mathcal{S}(t)$.

We denote the monoid of the terms by $\mathcal{T}(\mathcal{C}, \mathcal{V}, \mathcal{F})$ (or simply $\mathcal{T}$) and the monoid of the parameterized terms by $\mathcal{T}(\mathcal{C}, {}^{\#}\mathcal{V}, \mathcal{F})$ (or ${}^{\#}\mathcal{T}$). A term without function names is passive. We denote the set of all passive terms by $\mathcal{P}(\mathcal{C}, \mathcal{V})$. Let $\mathcal{G}(\mathcal{T}) \subset \mathcal{T}(\mathcal{C}, \mathcal{V}, \mathcal{F})$ be the set of ground terms, i.e. terms without variables. Let $\mathcal{O}(\mathcal{T}) \subset \mathcal{G}(\mathcal{T})$ be the set of object terms, i.e. ground passive terms.

[3]The left one and the right one.

Multiplicity of $h \in \mathcal{T} \setminus \{\lambda\}$ in a term $t$ is the number of occurrences of $h$ in $t$. Let us denote it by $\mu_h(t)$. The length $ln(t)$ of a term $t$ is defined by induction: $ln(\lambda) = 0$; if $t \in \mathcal{C} \cup \mathcal{V}$, then $ln(t) = 1$; $ln(f(t_1, \ldots, t_n)) = 1$; otherwise $t = t_1 \,\texttt{++}\, t_2$, where $t_i \neq \lambda$, and $ln(t) = ln(t_1) + ln(t_2)$.

Given a subset of the variables $\mathcal{V}_1 = \mathcal{S}_1 \cup \mathcal{E}_1$ where $\mathcal{S}_1 \subset \mathcal{S}$, $\mathcal{E}_1 \subset \mathcal{E}$, a substitution is a mapping $\theta : \mathcal{V}_1 \to \mathcal{T}(\mathcal{C}, \mathcal{V}, \mathcal{F})$ such that $\theta(\mathcal{S}_1) \subset \mathcal{C} \cup \mathcal{S}$. A substitution can be extended to act on all terms homomorphically. A substitution is called *ground*, *object*, or *strict* iff its range is a subset of $\mathcal{G}(\mathcal{T})$, $\mathcal{O}(\mathcal{T})$ or $\mathcal{T}(\mathcal{V})$ (i.e. passive terms), respectively. We use notation $s = t\theta$ for $s = \theta(t)$, call $s$ an *instance* of $t$ and denote this fact by $s \ll t$.

Formal replacement $\mathcal{V}$ with ${}^{\#}\mathcal{V}$, $\mathcal{S}$ with ${}^{\#}\mathcal{S}$, $\mathcal{E}$ with ${}^{\#}\mathcal{E}$ gives analogous notations for the parameterized terms.

The following two syntax constructions represent the same string of characters: `'a' ++ 'b' ++ 'c'` and `'abc'`. I.e. the second is a shortcut for the first. Examples of the variables are `s.rA`, `e.cls`, $\texttt{e.A}_5$, $\texttt{s.x}_{[*]}$. I.e. the variable's names may be identifiers possibly indexed. Examples of the parameters are `#s.rA`, $\texttt{\#e.p}_{(\tau[i],c[*])}$. The indices will be freely used and sometimes bring semantics meaning, which will be explained in corresponding examples usage.

A program $P$ in $\mathcal{L}$ is a pair $\langle t, R\rangle$, where $t$ is a parameterized term called *initial* and $R$ is a finite set of rules of the form $f(p_1, \ldots, p_k) = r$, where $f \in \mathcal{F}_k$, for each $(1 \leq i \leq k)$, $p_i$ is a passive term, the term $r$ may include some function names from $R$, $\mathcal{V}(r) \subseteq \mathcal{V}(f(p_1, \ldots, p_k))$. The variables in $\mathcal{L}$ cannot be subscripted by the indices including the asterisk sign.

The following program $\langle t, R\rangle$ is a predicate checking whether two given input object terms are equal.

**Example 1.** *$t$ is* `eq(#e.p, #e.q)` *and $R$ is*
```
eq(s.x ++ e.xs, s.x ++ e.ys) = eq(e.xs, e.ys);
eq(λ, λ) = 'T';
eq(e.xs, e.ys) = 'F';
```

The syntax of programs in $\mathcal{L}_*$ is defined similarly to the $\mathcal{L}$-program syntax. The only difference is: the variables in $\mathcal{L}_*$ should be subscripted by the indices including the asterisk sign. Henceforth we use the dotted equals sign $\doteq$ to denote textual (syntactic) identity. The sign `:=` denotes an assignment.

## 2.2 On Semantics of the Pattern Matching

Associativity of the concatenation creates a problem with the pattern matching, namely, given a term $\tau$ and a rule $(l = r) \in R$, then there can be several substitutions matching $\tau$ against $l$. Thus we here have a kind of non-determinism. An example is as follows:

**Example 2.** *$\tau =$ `f('abcabc', 'bc')` and $l =$ `f(e.x ++ e.w ++ e.y, e.w)`. There exist two substitutions matching the terms: the first one is $\theta_1(e.x) =$ `'a'`$, \theta_1(e.w) =$ `'bc'`$, \theta_1(e.y) =$ `'abc'`, the second one is $\theta_2(e.x) =$ `'abca'`$, \theta_2(e.w) =$ `'bc'`$, \theta_2(e.y) = \lambda$.*

To make the pattern matching unambiguous in the language $\mathcal{L}$, we take the following decision arisen from Markov's normal algorithms [13] and used in Refal [20]: (1) if there is more than one way of assigning values to the variables in the left-side of a rule in order to achieve matching, then the one is chosen in which the leftmost e-variable takes the shortest value; (2) if such a choice still gives more than one substitution, then the chosen e-variable shortest value is fixed and the case (1) is applied to the leftmost e-variable from the e-variables excluding considered ones, and so on while the whole list of the e-variables in the left-side of the rule is not exhausted.

In the sequel we refer to this rule as Markov's rule and to such a substitution as a Markov substitution on $l$, matching $\tau$. Given two terms $s, t$ and a Markov substitution $\theta$ such that $s = t\theta$, we call $s$ by a Markov instance and denote this fact by $s \lll t$.

**Example 3.** *$\tau =$ `f('abacad')` and $l =$ `f(`$e.x$ `++ 'a' ++` $e.y$ `++ 'a' ++` $e.z$`)`. There exist three substitutions matching the terms: the first one is $\theta_1(e.x) = \lambda, \theta_1(e.y) =$ `'b'`$, \theta_1(e.z) =$ `'cad'`; the second one is $\theta_2(e.x) = \lambda, \theta_2(e.y) =$ `'bac'`$, \theta_2(e.z) =$ `'d'`; the third one is $\theta_3(e.x) =$ `'ab'`$, \theta_3(e.y) =$ `'c'`$, \theta_3(e.z) =$ `'d'`.*

*The leftmost e-variable is e.x. Both in the first and the second substitutions the length of e.x's values is zero. The next leftmost e-variable is e.y and ln(`'b'`) < ln(`'bac'`). The first substitution meets Markov's rule.*

Given a term of the form `f(`$t_1, \ldots, t_n$`)` where for all $(1 \leq i \leq n)$, $t_i \in \mathcal{O}(\mathcal{T})$, and a term `f(`$p_1, \ldots, p_n$`)` where all $p_i$ are passive terms, matching `f(`$t_1, \ldots, t_n$`)` against `f(`$p_1, \ldots, p_n$`)` can be viewed as solving the following system of equations in the free monoid of the object terms $\mathcal{O}(\mathcal{T})$.

$$\begin{cases} p_1 & = & t_1 \\ & \ldots & \\ p_n & = & t_n \end{cases}$$

We look for all values of the variables (i.e. substitutions $\theta_i$) from $\mathcal{V}($`f(`$p_1, \ldots, p_n$`)`$)$ such that for each $i$ and each $(1 \leq j \leq n)$, $\theta_i(p_j) = t_j$ and if the values' set is not empty we choose: (1) in the case of the language $\mathcal{L}$, the only Markov substitution; (2) in the case of the language $\mathcal{L}_*$, nondeterministically one from the substitutions. In the second case the variables should be subscripted by the required indices. This system has an important property: all $t_i$ do not contain variables.

The following nondeterministic program $\langle t, R \rangle$ written in $\mathcal{L}_*$ results in one of the strings: $\lambda$, `'ab'`, `'F'`.

**Example 4.** *t is `f('abacad')` and R is*

`f(e.x`$_{[*]}$ `++ 'a' ++ e.y`$_{[*]}$ `++ 'a' ++ e.z`$_{[*]}$`) = e.x`$_{[*]}$`;`
`f(e.x`$_{[*]}$`) = 'F';`

## 2.3 Mutual Instances of the Terms

In this section we introduce a notion of a trivial substitution and write down the structure of the substitutions certificating the mutual instantiation of two given terms.

The associative concatenation causes mutual instances of the terms. For example, both $e.x \ll e.y$ `++` $e.z$ and $e.y$ `++` $e.z \ll e.x$ hold. That motivates the following notation.

**Definition 1.** *Let $\mathcal{V}_1$ and $\mathcal{V}_2$ be two sets of the variables such that $\mathcal{V}_1 \cap \mathcal{V}_2 = \emptyset$, $\mathcal{V}_i \subset \mathcal{V}$. Let $\mathcal{S}_i = \mathcal{V}_i \cap \mathcal{S}$ and $\mathcal{E}_i = \mathcal{V}_i \cap \mathcal{E}$. A substitution $\theta : \mathcal{V}_1 \to \mathcal{T}(\mathcal{V}_2)$ is called trivial iff (1) for all distinct $x, y \in \mathcal{V}_1$, $\mathcal{V}_2(\theta(x)) \cap \mathcal{V}_2(\theta(y)) = \emptyset$; (2) for each $x \in \mathcal{E}_1$, $\theta(x)$ is concatenation of certain* distinct *e-variables, possibly none of them and at most with one occurrence of each e-variable; (3) $\theta(\mathcal{S}_1) \subset \mathcal{S}_2$.*

Clearly, given a term $t$, there is at most finitely many both substitutions $\theta$ and terms $\tau$ such that $t \ll \tau\theta$, modulo variables' renaming, and for each $v \in \mathcal{V}(\tau)$, $ln(\theta(v)) > 0$, since such a substitution takes finitely many $t$'s subterms as its values.

Suppose $\mathcal{V}_1 = \{s.u_1, s.v_1, e.x_1\}$ and $\mathcal{V}_2 = \{s.u_2, s.v_2, s.w_2, e.x_2, e.y_2\}$, then the substitution $\theta(s.u_1) = s.u_2$, $\theta(s.v_1) = s.v_2$, $\theta(e.x_1) = e.x_2$ `++` $e.y_2$ is trivial, while the following two substitutions $\eta_1(s.u_1) = s.v_2$, $\eta_1(s.v_1) = s.v_2$, $\eta_1(e.x_1) = e.x_2 \mathtt{++} e.y_2$ and $\eta_2(s.u_1) = s.u_2$, $\eta_2(s.v_1) = s.v_2$, $\eta_2(e.x_1) = e.x_2 \mathtt{++} s.w_2 \mathtt{++} e.y_2$ are not. For the sake of the paper length limitations we leave the proof of the following simple proposition to the reader.

**Proposition 1.** *Let $t_1$, $t_2$ be two arbitrary terms. Both $t_1 \ll t_2$ and $t_2 \ll t_1$ hold iff either $t_1$ equals to $t_2$ modulo variables' names, or all substitutions $\theta$, $\eta$ such that $t_1\theta = t_2$ and $t_1 = t_2\eta$ are trivial.*

## 3 Examples Related to the Associative Concatenation

In the supercompilation method the meta-step `MStep` discussed in Introduction is called a *driving*. In this section we are concerned with specific problems of the driving, arisen from the associative concatenation. We study in detail important examples aiming to clarify the reader intuition on the driving over the strings.

Let a program $\langle t, R\rangle$ be given. Let $g$ (a *guard*) be a predicate depending on parameters from ${}^{\#}\mathcal{V}(t)$. The guard is defined by the following grammar:
$g$ `::= 'T' |` $qs$
$qs$ `::=` $cn$ `|` $cn \wedge qs$
$cn$ `::=` $\neg(p = d)$
where $p \in {}^{\#}\mathcal{V}(t)$, $d$ is a parameterized term. We use ${}^{\#}\mathcal{V}(g)$ to denote the set of parameters in the guard $g$ – similarly to the one used for the terms.

The driving `drv(t,g,R)` takes these three input arguments and results in a tree being branched with the input parameters from ${}^{\#}\mathcal{V}(t)$. In general (in supercompilation), `t` is an arbitrary parameterized term including some function names from the given program. Informally speaking, from a semantics point of view, `t` can be seen as a kind of a *goal* used in Prolog. For the sake of simplicity, in this paper we consider only parameterized initial terms of the following kind $\mathtt{f}(q_1, \ldots, q_n)$, where for each $(1 \le i \le n)$, $q_i \in \mathcal{P}({}^{\#}\mathcal{V})$. The tree root is labeled by the pair $\langle t, g\rangle$. Each other node is labeled by a pair $\langle p, h\rangle$, where $p$ is a parameterized term, $h$ is a guard imposed on parameters from ${}^{\#}\mathcal{V}(p)$. Given a node $\langle t, h\rangle$, the edges originating from this node are labeled by predicates on the parameters from ${}^{\#}\mathcal{V}(t) \cup {}^{\#}\mathcal{V}(h)$.

The tree generated by `drv(t,g,R)` is finite in the case when it is assumed that evaluation of `drv(t,g,R)` takes a time uniformly bounded by the size of the input parameters, otherwise the tree may be potentially infinite. We now turn ourselves to the case $g =$ `'T'`; some remarks on the general case will be given below.

The algorithm `Step` (see Section 1) consists of two subroutines: the pattern matching, choosing a rewriting rule and generating a substitution $\theta : \mathcal{V} \to \mathcal{O}(\mathcal{T})$, and replacement of the variables in the right-side of this rule by the generated substitution.

Consequently the driving should include meta-extension of these subroutines. The corresponding meta-extension of the pattern matching is a kind of unification used in Prolog and is described below in details. It results in a tree with leaves labeled by substitutions of the following kind $\theta : \mathcal{V} \to \mathcal{P}({}^{\#}\mathcal{V})$. Given these substitutions the second meta-subroutine applies them to the right-side of the rewriting rule being considered and it labels the corresponding leaves by the substations' result.

Given a parameterized term $\mathtt{f}(t_1, \ldots, t_n)$ where for each $(1 \le i \le n)$, $t_i \in \mathcal{P}({}^{\#}\mathcal{V})$, and a term $\mathtt{f}(p_1, \ldots, p_n)$ where all $p_i \in \mathcal{P}(\mathcal{V})$, the extended matching of $\mathtt{f}(t_1, \ldots, t_n)$ against $\mathtt{f}(p_1, \ldots, p_n)$

can be seen as solving of the following systems of *parameterized* equations.

$$\begin{cases} p_1 & = & t_1 \\ & \dots & \\ p_n & = & t_n \end{cases}$$

The following two examples are crucial.

**Example 5.**
*Consider the following program $P = \langle \tau, R \rangle$, $\tau =$ `f('a' ++ #e.p, #e.p ++ 'a')` and $R$ is*

```
f(e.x, e.x) = 'T';
f(e.x, e.y) = 'F';
```

*The root of the tree generated by* `drv(τ,'T',R)` *is $\langle \tau$, `'T'`$\rangle$. The extended pattern matching has to solve the following system of the parameterized equations:*

$$\begin{cases} \texttt{e.x} & = & \texttt{'a' ++ \#e.p} \\ \\ \texttt{e.x} & = & \texttt{\#e.p ++ 'a'} \end{cases} \qquad (\star)$$

*It is equivalent to the only relation $\Phi($`#e.p`$)$ (equation) – `'a' ++ #e.p` $=$ `#e.p ++ 'a'` imposed on the parameter* `#e.p`*.*[4]

*At naïve glance, $\Phi($`#e.p`$)$ must be the predicate labeling the first branch outcoming from the root, while the second branch must be labeled by the negation of the predicate $\neg\Phi($`#e.p`$)$. $\Phi($`#e.p`$)$ narrows the range of* `#e.p`*. But the problem is $\Phi($`#e.p`$)$ cannot be represented in the pattern language, i.e. using at most finitely many of the patterns to define a one-step program (a result of* `drv(τ,'T',R)`*). A recursion should be used to check whether a given input data belongs to the truth set of $\Phi($`#e.p`$)$. This recursion is unfolded as an infinite tree.*

*The multiplicity $\mu_{e.x}($`f(e.x, e.x)`$) > 1$ causes this problem: the system $(\star)$ implies an equation in which the parameter* `#e.p` *plays a role of a variable and* both sides of the equation contain `#e.p`.

Henceforth, we basing on Example 5 impose an additional restriction on the language $\mathcal{L}$: given a program $P = \langle \tau, R \rangle$ we require for all `(l = r)` $\in R$ and for all $v \in \mathcal{E}($`l`$)$, $\mu_v($`l`$) = 1$. Notice that we do not impose any restriction on uses of the s-variables.

**Example 6.** *The following program demonstrates another problem.*
*$P = \langle \tau, R \rangle$, $\tau =$* `f(#e.p ++ 'a' ++ #e.q)` *and $R$ is as follows.*

```
f(e.x ++ 'a' ++ e.y) = e.x ++ 'b' ++ e.y;
f(e.x) = e.x;
```

*The root of the tree generated by* `drv(τ,'T',R)` *is $\langle \tau$, `'T'`$\rangle$. The extended pattern-matching has to solve the following parameterized equation:*

$$\texttt{e.x ++ 'a' ++ e.y} = \texttt{\#e.p ++ 'a' ++ \#e.q}$$

*That is to say the substitutions reducing the equation to identities have to be expressed through the parameters and we are only interested in the substitutions taking into account the semantics of the pattern (the left-side): the value of* `e.x` *cannot contain* `'a'`*. The character* `'a'` *on the right-side of the equation is allowed to take any position in this unknown string.*

[4]It is easy to see that its solution set is $\{\theta_i($`#e.p`$) =$ `'a`$^i$`'` $\mid i \in \mathbb{N}\}$.

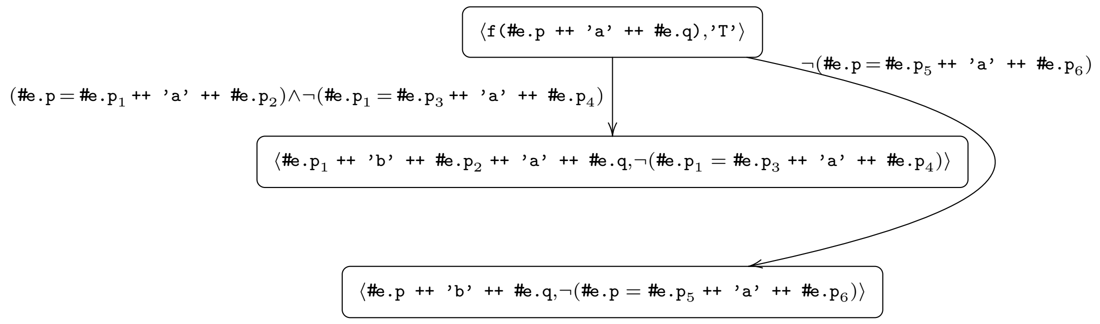
⟨f(#e.p ++ 'a' ++ #e.q),'T'⟩
(#e.p = #e.p$_1$ ++ 'a' ++ #e.p$_2$)∧¬(#e.p$_1$ = #e.p$_3$ ++ 'a' ++ #e.p$_4$)
¬(#e.p = #e.p$_5$ ++ 'a' ++ #e.p$_6$)
⟨#e.p$_1$ ++ 'b' ++ #e.p$_2$ ++ 'a' ++ #e.q,¬(#e.p$_1$ = #e.p$_3$ ++ 'a' ++ #e.p$_4$)⟩
⟨#e.p ++ 'b' ++ #e.q,¬(#e.p = #e.p$_5$ ++ 'a' ++ #e.p$_6$)⟩

Figure 1: The result of the driving `drv(`$\tau$`,'T',`$R$`)` from Example 6

*Let us solve this equation. In the case* `#e.p` *contains the characters* `'a'`*, we choose the first occurrence of* `'a'` *and split the unknown data* `#e.p` *by this occurrence as follows* #e.p$_1$ ++ 'a' ++ #e.p$_2$*, thinking about the representation as a pattern where the parameters* #e.p$_1$*,* #e.p$_2$ *take a role of the variables. That implies* e.x = #e.p$_1$*, where* `'a'` *does not occur in* #e.p$_1$*, and our equation can be reduced to* e.y = #e.p$_2$ ++ 'a' ++ #e.q*. In the case* `#e.p` *contains no* `'a'`*, we conclude that* `e.x = #e.p` *and* `e.y = #e.q`*.*

*In the terms of the language $\mathcal{L}$ this solution set and the result of the corresponding substitutions in the right-sides of $R$ can (modulo the variables) be described by the folllowing program $P_1 = \langle \tau_1, R_1 \rangle$, where $\tau_1 = f_1$*`(#e.p, #e.q)` *and $R_1$ is as follows.*

$f_1$(#e.p$_1$ ++ 'a' ++ #e.p$_2$, #e.q) = #e.p$_1$ ++ 'b' ++ #e.p$_2$ ++ 'a' ++ #e.q;
$f_1$(#e.p, #e.q) = #e.p ++ 'b' ++ #e.q;

*Removing the sharp signs in $R_1$, we now may obtain a program written in $\mathcal{L}$. $P_1$ is almost a textual representation of the result tree $T$ generated by* `drv(`$\tau$`,'T',`$R$`)` *and given in Figure 1.*

Example 6 demonstrates that the unification in $\mathcal{L}$ may be ambiguous and motivates the splitting transformation given in the following sections.

**Example 7.** *Consider the following program $P = \langle \tau, R \rangle$, where $R$ is*

```
f(s.x ++ s.x) = 'T';
f(e.z) = 'F';
```

*Suppose the initial term $\tau$ =* `f(#s.p ++ #s.q)`*, then the extended pattern-matching has to solve the following parameterized equation:* `s.x ++ s.x = #s.p ++ #s.q;`*. It is equivalent to the equation* `#s.p = #s.q`*. The driving* `drv(`$\tau$`,'T',`$R$`)` *will produce $P_1 = \langle f_1$*`(#s.p,#s.q)`*$, R_1 \rangle$, where $R_1$ is*

$f_1$(#s.p, #s.p) = 'T';
$f_1$(#s.p, #s.q) = 'F';

*In the case $\tau$ =* `f(#s.p ++ 'a')` *we have the equation:* `s.x ++ s.x = #s.p ++ 'a';`*. It is equivalent to* `#s.p = 'a'`*. The driving* `drv(`$\tau$`,'T',`$R$`)` *produces $P_2 = \langle f_2$*`(#s.p)`*$, R_2 \rangle$, where $R_2$ is*

$f_2$('a') = 'T';
$f_2$(#s.p) = 'F';

*The left-sides of the first rewriting rules reflect the relations generated on the s-parameters. Analogously, if $p$ is a pattern in a program and $v \in \mathcal{S}(p), \mu_v(p) = k$, the extended matching may produce an $n$-ary relation imposed on the s-parameters from the initial term, where $(n \leq k)$ depends on the initial term. For example,* `drv(f(#s.p,#s.p),'T',R)` *generates the trivial relation (i.e. no relation at all).*

## 4 The Driving over Strings

In this section we formulate and prove two constructive theorems being our main contribution in this paper. The first theorem concerns the nondeterministic extended pattern-matching in the language $\mathcal{L}_*$. The second theorem deals with Markov's rule leading to the deterministic choice of the matching substitution (Section 2.2), and it is more subtle. Both the theorems are of interest to the theory of the equations in the free monoid.

### 4.1 Driving for One Rule Programs

Here we explore the extended pattern-matching for one rule programs written in a syntactic subset of both $\mathcal{L}_*$ and $\mathcal{L}$ languages. The subset is characterized by the following restriction imposed on the left-sides of the rules: given a rule, each e-variable may occur in the left-side of the rule at most once. Let $\mathcal{M}_*$ denote such a subset of $\mathcal{L}_*$ and $\mathcal{M}$ denote the similar subset of $\mathcal{L}$. The corresponding algorithms are very similar to the Prolog unification. The main difference is the ambiguity originating from the associative concatenation.

We describe a program transformation splitting a given one-rule program $P$ in the language $\mathcal{M}_*$ (or $\mathcal{M}$): it creates a program $P_1$ with a number of rules, such that $P_1$ is semantics equivalent to $P$ up to a simple syntactic transformation modifying the initial-term arity of $P$, which will be clear from the context.

**Definition 2.** *Let $(u = q)$ be a parameterized equation (a predicate) such that $u \in {}^{\#}\mathcal{V}$, $q \in \mathcal{T}({}^{\#}\mathcal{V})$. We say the substitution defined by the assignment $(u := q)$ is conjugated of the predicate $(u = q)$.*

**Definition 3.** *Let $t$ be a parameterized term and $\pi$ be a predicate $\bigwedge_{i=1}^{n}(u_i = q_i)$ such that for each $(1 \leq i \leq n)$, $q_i \in \mathcal{T}({}^{\#}\mathcal{V})$, $u_i \in {}^{\#}\mathcal{V}(t) \cup \{\bigcup_{j=1}^{i-1} {}^{\#}\mathcal{V}(q_j)\}$; the predicates $(u_i = q_i)$ are ordered by their indices and for each $(1 \leq i \leq n)$, $q_i$ may share variables with $u_1, \ldots, u_{i-1}$. $\pi$ is called a narrowing of the parameters from ${}^{\#}\mathcal{V}(t)$ (or of the term $t$). Let $K$ be either the language $\mathcal{L}$ or $\mathcal{L}_*$ (see Section 2.1). We say $\pi$ is written in $K$ iff for each $i$, $q_i$ is written in $K$.*

*Let $\zeta_i$ denote the substitution conjugated of $(u_i = q_i)$. We denote the narrowing $\pi$ by the sequence $\zeta_1; \ldots \zeta_n$, stressing the ordering, where the semicolon sign stands both for conjunction and formal composition due to sharing of the variables.*

*The substitution $\theta(u_1) = q_1\zeta_2 \ldots \zeta_n$; $\ldots$ $\theta(u_i) = q_i\zeta_{i+1} \ldots \zeta_n$; $\ldots$ $\theta(u_{n-1}) = q_{n-1}\zeta_n$; $\theta(u_n) = q_n$; is conjugated of the predicate $\pi$ and denoted by $\tilde{\pi}$. The following narrowing $(u_1 = q_1\zeta_2 \ldots \zeta_n)$; $\ldots$ $(u_{n-1} = q_{n-1}\zeta_n)$; $(u_n = q_n)$ is called a formal normal form of the predicate $\pi$ and denoted by $\breve{\pi}$. We say $\breve{\pi}$ is obtained from $\pi$ by a formal composition.*

*Remark* 4.1.1. The conjuncts of the formal normal form of a narrowing may still share their variables. For example, $(\texttt{e.y} = \texttt{s.x}\texttt{++}\texttt{e.z})$; $(\texttt{s.u} = \texttt{s.x})$.

*Remark* 4.1.2. Given a narrowing $\pi$ written in $\mathcal{L}$, the relations defined by $\pi$ and $\breve{\pi}$ may be different. In the case the semicolon sign is understood purely as the conjunction, the following two narrowings have distinct satisfiability sets: $\pi$ is $(\texttt{e.y} = \texttt{e.z}\texttt{++}\texttt{'c'}\texttt{++}\texttt{e.p})$; $(\texttt{e.p} = \lambda)$ and

$\breve{\pi}$ is $(\mathtt{e.y} = \mathtt{e.z} \mathbin{++} \mathtt{'c'})$; $(\mathtt{e.p} = \lambda)$. In $\pi$, by the $\mathcal{L}$-semantics, the variable `e.z` cannot take on a string including the character `'c'`, but the same variable in $\breve{\pi}$ takes on the string `'c'` when `e.y` takes on `'cc'`. Another example: $\pi$ is $(\mathtt{e.y} = \mathtt{e.z} \mathbin{++} \mathtt{'c'} \mathbin{++} \mathtt{e.p})$; $(\mathtt{e.p} = \mathtt{'d'} \mathbin{++} \mathtt{e.p}_1)$ and $\breve{\pi}$ is $(\mathtt{e.y} = \mathtt{e.z} \mathbin{++} \mathtt{'cd'} \mathbin{++} \mathtt{e.p}_1)$; $(\mathtt{e.p} = \mathtt{'d'} \mathbin{++} \mathtt{e.p}_1)$. Given `'ccd'` as a value of `e.y`, in $\pi$ `e.z` takes on $\lambda$, while in $\breve{\pi}$ it takes on `'c'`.[5]

**Definition 4.** *We say $t \in \mathcal{T}(^{\#}\mathcal{V})$ matches against $p \in \mathcal{T}(\mathcal{V})$ iff there exists a substitution $\theta : \mathcal{V}(p) \to {}^{\#}\mathcal{V}$ such that $p\theta = t$.[6] We say $t$ is able to be matched against $p$ iff there exist a narrowing $\pi$ of $t$ and a substitution $\theta : \mathcal{V}(p) \to \mathcal{T}(^{\#}\mathcal{V}(\Im(\tilde{\pi})))$ such that $p\theta = t\tilde{\pi}$, where $\Im(\tilde{\pi})$ is the image of $\tilde{\pi}$. $\pi$ is called the narrowing of $t$ w.r.t. $p$.*

*A narrowing $\pi$ is called a most general narrowing of $t$ w.r.t. $p$ iff for each narrowing $\chi$ of $t$ w.r.t. $p$, $t\tilde{\pi} \ll t\tilde{\chi}$ implies that $\tilde{\chi}$ is a trivial substitution defined on $^{\#}\mathcal{V}(t\tilde{\pi})$ or $t\tilde{\chi}$ equals to $t\tilde{\pi}$ modulo parameter's names.*

Suppose $p \in \mathcal{T}(\mathcal{V})$, $t \in \mathcal{T}(^{\#}\mathcal{V})$, there may be a number of pairs – a *most general* narrowing and a substitution being able to match $t$ against $p$. There may be a number of substitutions matching $t$ against $p$. The following example and the examples given above show that.

Both $(\mathtt{\#e.u} = \mathtt{\#e.v} \mathbin{++} \mathtt{'c'} \mathbin{++} \mathtt{\#e.v}_1)$ and $(\mathtt{\#e.u} = \mathtt{\#e.v} \mathbin{++} \mathtt{'c'} \mathbin{++} \mathtt{\#e.v}_1 \mathbin{++} \mathtt{\#e.v}_2)$ are most general narrowings of `#e.u` w.r.t. $\mathtt{e.x} \mathbin{++} \mathtt{'c'} \mathbin{++} \mathtt{e.y}$.

### 4.1.1 Nondeterministic Case

Given $p \in \mathcal{P}(\mathcal{V})$, $t \in \mathcal{P}(^{\#}\mathcal{V})$, let $\pi(p,t)$ denote a most general narrowing of $t$ with respect to $p$ and $\theta(p,t)$ denote a substitution matching the result $t\tilde{\pi}$ of this narrowing against $p$. Let $\langle \pi, \theta \rangle(p,t)$ denote this pair and $\Pi(p,t)$ denote the set of the most general narrowings of $t$ with respect to $p$. In the following theorem we do not impose any restriction on the parameterized term $t$. The proof below is given by induction on the length of $p$ and can be seen as an outline of an algorithm computing $\langle \Pi, \Theta \rangle(p,t)$.

Given $\langle \pi, \theta \rangle(p_1, t_1)$ and $\langle \Pi, \Theta \rangle(p,t)$, the binary operation $\oplus$ is defined as follows:
$\langle \pi_1, \theta_1 \rangle(p_1, t_1) \oplus \langle \Pi, \Theta \rangle(p,t) ::= \{\langle \pi_1; \pi,\ \theta_1; \theta \rangle(p_1 \mathbin{++} p, t_1 \mathbin{++} t) \mid \langle \pi, \theta \rangle(p,t) \in \langle \Pi, \Theta \rangle(p,t)\}$, where the semicolon sign is used for the concatenation meaning a composition.

**Theorem 1.** *For any $p \in \mathcal{P}(\mathcal{V})$, $t \in \mathcal{P}(^{\#}\mathcal{V})$ such that for all $v \in \mathcal{E}(p)$, $\mu_v(p) \le 1$, there exists a finite set $\Pi(p,t)$ s.t. each $\pi \in \Pi(p,t)$ written in $\mathcal{L}_*$ and for each, written in $\mathcal{L}_*$, narrowing $\phi$ of $t$ w.r.t. $p$ there exists $\pi \in \Pi(p,t)$ such that $t\tilde{\phi} \ll t\tilde{\pi}$.*

*Proof.* Starting with $\langle \pi, \theta \rangle(p,t) := \langle \lambda, \lambda \rangle$, we will sequentially add conjuncts to the definition of $\pi(p,t)$ and assignments to the definition of $\theta(p,t)$. By default, we assume that such a narrowing exists, otherwise we explicitly indicate the opposite. $\langle \Pi, \Theta \rangle(p,t)$ stands for the set of the generated $\langle \pi, \theta \rangle(p,t)$. Its initial value is the empty set. Let $c$ possibly subscripted denote a character.

**The base case 1:** $ln(p) = 0$.

We have an equation $\lambda = t$ implying the following narrowings for all $v \in {}^{\#}\mathcal{E}(t)$, $\pi_v := (v = \lambda)$, and requiring $ln(t\tilde{\pi}_t)$ to be zero, where $\tilde{\pi}_t(v) = \lambda$. Let $\omega$ be the sequence of all $\langle \pi_v, \lambda \rangle$, then $\langle \pi, \theta \rangle(p,t) := \omega$; and $\langle \Pi, \Theta \rangle(p,t) := \{\langle \pi, \theta \rangle(p,t)\}$. If $ln(t\tilde{\pi}_t) > 0$, then $\langle \Pi, \Theta \rangle(p,t) := \emptyset$.

[5] We leave to the reader to prove that the formal composition is sound in the language $\mathcal{L}_*$ and, if $\pi$ is a narrowing $(u_1 = q_1); (u_2 = q_2)$ written in $\mathcal{L}$ s.t. $q_1$ is passive and for each $v \in \mathcal{E}(q_i)$, $\mu_v(q_i) \le 1$, then the formal composition is sound for $\pi$.

[6] That is to say, $t \ll p$ if the set $\mathcal{V} \cup {}^{\#}\mathcal{V}$ is considered as a variable set.

**The base case 2:** $ln(p) = 1$ and $p$ is $v \in \mathcal{E}$.

We have an equation $\texttt{e.x}_* = t$ implying no restriction on the parameters from $t$ and the only substitution $\texttt{e.x}_* := t$ defines its solution set. $\langle \pi, \theta \rangle(p,t) := \langle \lambda, (\texttt{e.x}_* := t) \rangle$ and $\langle \Pi, \Theta \rangle(p,t) := \{\langle \pi, \theta \rangle(p,t)\}$.

**Inductive step:** Assume that for all $q \in \mathcal{P}(\mathcal{V}), g \in \mathcal{P}(^{\#}\mathcal{V})$ such that $(\mu_v(q) \leq 1) \wedge (ln(q) \leq k)$ the proposition statement holds (let it be denoted by $\Phi(q,g)$) and the finite set $\langle \Pi, \Theta \rangle(q,g)$ was generated.

We want to prove that for all $p \in \mathcal{P}(\mathcal{V}), t \in \mathcal{P}(^{\#}\mathcal{V})$ such that $(\mu_v(p) \leq 1) \wedge (ln(p) = k+1)$, $\Phi(p,t)$ holds.

**(1):** *Suppose* $p \doteq p_1 \texttt{++}\, q$, *where* $p_1$ *is a character* $c_1$ *or* $v \in \mathcal{S}$.

<u>**(1.a)**</u> If $t \doteq t_{1a}$ ++ $g$, where $t_{1a}$ is a character $c_2$ or $t_{1a} \in {}^{\#}\mathcal{S}$, then the existence of a narrowing of $t$ w.r.t. $p$ implies the existence of a narrowing (maybe trivial) of $t_{1a}$ w.r.t. $p_1$. $\pi(c_1,c_2)$ is either the tautology ($\pi(c_1,c_1) = \lambda$) or the contradiction $\pi(c_1,c_2)$, $c_1 \neq c_2$. $\langle \pi, \theta \rangle(c_1, \texttt{\#s.u}_*) := \langle (\texttt{\#s.u}_* = c_1), \lambda \rangle$, $\langle \pi, \theta \rangle(\texttt{s.x}_*, c_2) := \langle \lambda, (\texttt{s.x}_* \texttt{ := } c_2) \rangle$, $\langle \pi, \theta \rangle(\texttt{s.x}_*, \texttt{\#s.u}_*) := \langle \lambda, (\texttt{s.x}_* \texttt{ := } \texttt{\#s.u}_*) \rangle$. Let $\langle \pi_{p_1}, \theta_{t_{1a}} \rangle$ denote the pair $\langle \pi, \theta \rangle(p_1, t_{1a})$ generated.

Given $\langle \pi, \theta \rangle(q, g\tilde{\pi}_{p_1}) \in \langle \Pi, \Theta \rangle(q, g\tilde{\pi}_{p_1})$, in the case $p_1 \doteq \texttt{s.x}_*$ the assignment $(\texttt{s.x}_* \texttt{ := } t_{1a})$ must be compatible with the substitution $\theta(q, g\tilde{\pi}_{p_1})$ generated by our inductive assumption. A clash of these two substitutions may appear only if $\mu_{\texttt{s.x}_*}(t) > 1$. In such a case there exists an assignment $(\texttt{s.x}_* \texttt{ := } t_{2a})$ in $\theta(q, g\tilde{\pi}_{p_1})$. The needed narrowing of $t_{1a}$ exists iff these two values of $\texttt{s.x}_*$ coincide. That gives a relation $\pi_{t12} := (t_{1a} = t_{2a})$, which may be a contradiction, a tautology, either $(\texttt{\#s.u}_* = \texttt{\#s.w}_*)$ or $(\texttt{\#s.u}_* = c)$ with $c \doteq t_{2a}$, or $(\texttt{\#s.w}_* = c_2)$.

Let $\omega$ denote $\langle \pi_{t12}, \lambda \rangle$ if $\mu_{\texttt{s.x}_*}(t) > 1$, and $\lambda$ otherwise. If a contradiction takes place, then $\langle \Pi, \Theta \rangle(p,t) := \emptyset$;. Otherwise, we assign $\langle \Pi, \Theta \rangle(p,t) := \langle \pi, \theta \rangle(p_1, t_{1a}) \oplus \langle \Pi, \Theta \rangle(q, g\tilde{\pi}_{p_1}) \oplus \omega$.

<u>**(1.b)**</u> If $t \doteq \lambda$, then we have the following contradiction: there is no narrowing of $t$ w.r.t. $p$. $\langle \Pi, \Theta \rangle(p,t) := \emptyset$;.

<u>**(1.c)**</u> If $t \doteq \texttt{\#e.w}_* \texttt{ ++ } g$, then the unknown term $\texttt{\#e.w}_*$ can be represented in one of the following forms $\lambda$ or $v$ ++ $\texttt{\#e.w}_{*1}$, where $v \in {}^{\#}\mathcal{S}$. These forms define the following two possible narrowings of $\texttt{\#e.w}_*$: $\pi_{11} := (\texttt{\#e.w}_* = \lambda)$, $\pi_{12} := (\texttt{\#e.w}_* = v \texttt{ ++ } \texttt{\#e.w}_{*1})$.

In the second case $t\tilde{\pi}_{12}$ is of the form considered in the case **(1.a)**. We compute $\langle \Pi, \Theta \rangle(p, t\tilde{\pi}_{12})$ following the case **(1.a)**.

In the case $\texttt{\#e.w}_* = \lambda$ we have to test $t\tilde{\pi}_{11}$ by all the three variants **(1.a)**, **(1.b)**, **(1.c)**. Notice that $ln(t\tilde{\pi}_{11}) < ln(t)$. And we deduce that the variant **(1.c)** can be involved by the case **(1)** at most finitely many times. We have $\langle \Pi, \Theta \rangle(p,t) = \bigcup_{i \in \{1,2\}} \{\langle \pi_{1i}, \lambda \rangle \oplus \langle \Pi, \Theta \rangle(p, t\tilde{\pi}_{1i})\}$.

**(2):** *Suppose* $p \doteq \texttt{e.y}_* \texttt{ ++ } q \texttt{ ++ } p_1$, *where* $p_1$ *is a character* $c_1$ *or* $v \in \mathcal{S}$.

We literally repeat the steps of this proof given in the case **(1)**, taking into account that in the case **(2)** the ending terms play the role of the leading terms of the case **(1)** and the inductive assumption is applied to the term $\texttt{e.y}_* \texttt{ ++ } q$ rather than to $q$. This case has been considered.

**(3):** *Suppose* $p \doteq \texttt{e.y}_* \texttt{ ++ } c \texttt{ ++ } q \texttt{ ++ } \texttt{e.z}_*$, *where* $c$ *is a character.*

If the multiplicity $\mu_c(t) > 0$, then there exist $t_{c[i]}, g_{c[i]} \in \mathcal{P}(^{\#}\mathcal{V})$ such that $t \doteq t_{c[i]} \texttt{ ++ } c_{[i]} \texttt{ ++ } g_{c[i]}$ where both $t_{c[i]}$ and $g_{c[i]}$ may be $\lambda$, and the indexed $c_{[i]}$ indicates the $i$-th occurrence of $c$ in $t$ (from the left to the right), while $t_{c[i]}$, $g_{c[i]}$ stand for its neighbors. The inequality $ln(p) > ln(q \texttt{ ++ } \texttt{e.z}_*)$, by the induction assumption, implies that $\langle \Pi, \Theta \rangle(q \texttt{ ++ } \texttt{e.z}_*, g_{c[i]})$ can be computed and $(\langle \lambda, (\texttt{e.y}_* := t_{c[i]}) \rangle \oplus \langle \Pi, \Theta \rangle(q \texttt{ ++ } \texttt{e.z}_*, g_{c[i]})) \subset \langle \Pi, \Theta \rangle(p,t)$.

Given $v \in {}^{\#}\mathcal{V}$, if the multiplicity $\mu_v(t) > 0$, then there exist $t_{v[i]}, g_{v[i]} \in \mathcal{P}(^{\#}\mathcal{V})$ such that $t \doteq t_{v[i]} \texttt{ ++ } v_{[i]} \texttt{ ++ } g_{v[i]}$.

Suppose $v$ is $\texttt{\#s.u}_*$, then let $\pi_v$ denote $(\texttt{\#s.u}_* = c)$. For each $(0 < i \leq \mu_v(t))$ the set $\langle \pi_v, (\texttt{e.y}_* := t_{v[i]}\tilde{\pi}_v) \rangle \oplus \langle \Pi, \Theta \rangle(q \texttt{ ++ } \texttt{e.z}_*, g_{v[i]}\tilde{\pi}_v)$ is a subset of $\langle \Pi, \Theta \rangle(p,t)$.

Suppose $v$ is $\texttt{\#e.w}_*$. The unknown string $\texttt{\#e.w}_*$ may include the character $c$. We represent such a relation by the following syntax $\texttt{\#e.w}_* = \texttt{\#e.w}_{(v[i],c[*])} \mathbin{++} c_{[*]} \mathbin{++} \texttt{\#e.r}_{(v[i],c[*])}$ (denoted by $\kappa_v$) meaning that $c$ may take any position in the unknown string $\texttt{\#e.w}_*$, where $\texttt{\#e.w}_{(v[i],c[*])}$, $\texttt{\#e.r}_{(v[i],c[*])}$ are fresh parameters. For each $(0 < i \le \mu_v(t))$ the set $\langle \kappa_v, (\texttt{e.y}_* := t_{v[i]}\tilde{\kappa}_v \mathbin{++} \texttt{\#e.w}_{(v[i],c[*])})\rangle \oplus \langle \Pi, \Theta\rangle(q \mathbin{++} \texttt{e.z}_*, \texttt{\#e.r}_{(v[i],c[*])} \mathbin{++} g_{v[i]}\tilde{\kappa}_v)$ is a subset of $\langle \Pi, \Theta\rangle(p,t)$.

We now are ready to compute the set $\langle \Pi, \Theta\rangle(p,t)$. It is as follows.

$$\begin{aligned}\langle \Pi, \Theta\rangle(p,t) = {} & \bigcup_{1\le i\le \mu_c(t)}(\langle \lambda, (\texttt{e.y}_* := t_{c[i]})\rangle \oplus \langle \Pi, \Theta\rangle(q \mathbin{++} \texttt{e.z}_*, g_{c[i]})) \cup \\ & \bigcup_{v\in {}^{\#}\mathcal{S}(t)}(\bigcup_{1\le i\le \mu_v(t)}(\langle \pi_v, (\texttt{e.y}_* := t_{v[i]}\tilde{\pi}_v)\rangle \oplus \langle \Pi, \Theta\rangle(q \mathbin{++} \texttt{e.z}_*, g_{v[i]}\tilde{\pi}_v))) \cup \\ & \bigcup_{v\in {}^{\#}\mathcal{E}(t)}(\bigcup_{1\le i\le \mu_v(t)}( \quad \langle \kappa_v, (\texttt{e.y}_* := t_{v[i]}\tilde{\kappa}_v \mathbin{++} \texttt{\#e.w}_{(v[i],c[*])})\rangle \\ & \qquad\qquad \oplus \langle \Pi, \Theta\rangle(q \mathbin{++} \texttt{e.z}_*, \texttt{\#e.r}_{(v[i],c[*])} \mathbin{++} g_{v[i]}\tilde{\kappa}_v)))\end{aligned}$$

The case **(3)** has been proved: the set $\langle \Pi, \Theta\rangle(p,t)$ is finite.

**(4):** *Suppose* $p \doteq \texttt{e.y}_* \mathbin{++} \texttt{s.x}_* \mathbin{++} q \mathbin{++} \texttt{e.z}_*$. Let $\tau$ denote $c \in \mathcal{C}$ or $v \in {}^{\#}\mathcal{S}$.

This case follows by arguments similar to the previous case, differing only by the following two details. If $\mu_{\texttt{s.x}_*}(p) = 1$, then this s-variable does not impose any narrowing on a given or unknown character $\tau$, producing the only assignment $\langle \lambda, (\texttt{s.x}_* := \tau)\rangle$. If $\mu_{\texttt{s.x}_*}(p) > 1$, then each $\langle \pi, \theta\rangle(q \mathbin{++} \texttt{e.z}_*, g_{\tau[i]})$ includes a pair of the kind $\langle \rho, (\texttt{s.x}_* := \tau_2)\rangle$, where $\tau_2 \in \mathcal{C} \cup {}^{\#}\mathcal{S}$, and we have to take into account the narrowing which corresponds to the relation $(\tau = \tau_2)$.

The reader being not interested in all the details of the algorithm computing the set $\langle \Pi, \Theta\rangle(p,t)$ may skip to the case **(5)**. The concrete details are given in Appendix A.

The only possibility we have not yet considered is the following.

**(5):** *Suppose* $p \doteq \texttt{e.y}_* \mathbin{++} v \mathbin{++} q$, *where* $v \in \mathcal{E}$.

Since $\mu_{\texttt{e.y}_*}(p) = \mu_v(p) = 1$ we deal with this case as follows. We replace the term $\texttt{e.y}_* \mathbin{++} v$ with a fresh variable $\texttt{e.y}_{*1}$. By the induction assumption, one can compute $\mathcal{R} = \langle \Pi, \Theta\rangle(\texttt{e.y}_{*1} \mathbin{++} q, t)$ since $ln(p) > ln(\texttt{e.y}_{*1} \mathbin{++} q)$. For each sequence $\rho = \langle \pi, \theta\rangle(\texttt{e.y}_{*1} \mathbin{++} q, t) \in \mathcal{R}$, we replace $\langle \pi_{t_1}, (\texttt{e.y}_{*1} := t_1)\rangle$ by $\langle \pi_{t_1}, (\texttt{e.y}_* := t_{\varsigma 11})\rangle$; $\langle \lambda, (v := t_{\varsigma 12})\rangle$, where $\varsigma$ denotes an arbitrary splitting of $t_1 = t_{\varsigma 11} \mathbin{++} t_{\varsigma 12}$, including all semantics splittings similarly to the case **(4)**. Denote the obtained sequence by $\rho_\varsigma$. Denote the union of $\rho_\varsigma$ over all $\varsigma$ by $\mathcal{R}_\rho$. The set $\langle \Pi, \Theta\rangle(p,t)$ is the union of $\mathcal{R}_\rho$ over all $\rho$. The current case is considered.

The inductive step has been proved. This completes the proof. □

*Remark* 4.1.3. Both the definition of the narrowing and the proof of Theorem 1 expose duality of the narrowing and the substitution. These notations are inverted one to the other.

*Remark* 4.1.4. Given an equation $p = t$ where $p \in \mathcal{P}(\mathcal{V})$, $t \in \mathcal{P}({}^{\#}\mathcal{V})$ s.t. for all $v \in \mathcal{E}(p)$, $\mu_v(p) = 1$, its solution set is described by $\{\langle \tilde{\pi}, \theta\rangle \mid \langle \pi, \theta\rangle(p,t) \in \langle \Pi, \Theta\rangle(p,t)\}$.

*Remark* 4.1.5. In 1977 G. S. Makanin presented a nontrivial decision algorithm solving the satisfiability problem for word equations [12] (see also [4]). In fact, Makanin's algorithm is a semidecision procedure which enumerates solutions of word equations with constants. In the paper [9] Jaffar described Makanin's algorithm in explicit (transparent) terms. His procedure generates, given a word equation, a minimal and complete set of unifiers describing the set of all solutions. It stops if this set is finite. There are a number of simple algorithms dealing with specific classes of the equations in the free monoid. Given a word equation, they describe, in a certain way, the corresponding solution set. For example some of them generate finite graphs describing this set. Here the major problem is to create a formal description language which is both clear in details and allowing us to represent some infinite solution sets in finite terms. For instance any word equation itself is a description of its solution set, but, as a rule, such a description is very obscure. As far as we know, there exist no papers using the nondeterministic pattern matching as such a language and none considered the class of the

equations characterized in Theorem 1, namely the variables separated by the two equation sides such that at least one of the sides has at most one occurrence of each word variable.[7]
*Remark* 4.1.6. Of interest to us are the s-variables: they allow us to compare unknown characters and to work with an unknown alphabet.

**The nondeterministic driving.** Recall that $\mathcal{M}_*$ denotes a subset of $\mathcal{L}_*$ such that for any program $P \in \mathcal{M}_*$, for each rule `(l = r;)` from $P$ and for each $v \in \mathcal{E}(l)$, $\mu_v(l) = 1$ holds. The constructive proof of Theorem 1 is the principal part of the driving algorithm for one-rule programs written in $\mathcal{M}_*$. We denote the proof by $\Phi_1(\wp)$ where $\wp$ is the set $\langle \Pi, \Theta \rangle$ being transformed by the proof. So the computation included in the proof may be represented by the assignment $\wp_2 := \Phi_1(\wp_1)$ meaning that $\Phi_1$ takes the initial set $\wp_1 = \langle \lambda, \lambda \rangle$ as an input argument and results in $\wp_2$.

Let a program $P = \langle \tau, R \rangle$ written in $\mathcal{M}_*$ be given, where $\tau = \mathtt{f(t_1,\ldots,t_n)}$ and $R$ is $\mathtt{f(p_1,\ldots,p_n)\ =\ r;}$. Recall that the patterns $\mathtt{p_1,\ldots,p_n}$ may share some s-variables. Let $\mathtt{g_1,\ldots,g_m}$ be a certain enumeration of the set ${}^{\#}\mathcal{V}(\tau)$. The driving algorithm $\mathtt{drv}(\tau,\mathtt{'T'},R)$ works as follows: (1) $\wp_1 := \langle \lambda, \lambda \rangle$; (2) for each $(1 < i \le n)$, $\wp_i := \Phi_1(\wp_{i-1})$; (3) Given $\langle \pi, \theta \rangle \in \wp_n$, let for each $(1 \le i \le m)$, $\mathtt{q}_i$ denotes the right-side of the narrowing imposed on the parameter $\mathtt{g}_i$ in $\breve{\pi}$[8], if such a narrowing imposed on $\mathtt{g}_i$ exists; otherwise (i.e. the narrowing imposed on $\mathtt{g}_i$ is a tautology) $\mathtt{q}_i \doteq \mathtt{g}_i$. Let $R_1$ be the union of the rules of the following kind $\mathtt{f_2(q_1,\ldots,\ q_m)\ =\ r}\theta\mathtt{;}$ over all $\langle \pi, \theta \rangle \in \wp_n$. Let $\tau_2$ be $\mathtt{f_2(g_1,\ldots,\ g_m)}$.

The algorithm $\mathtt{drv}(\tau,\mathtt{'T'},R)$ results in the program $P_2 = \langle \tau_2, R_2 \rangle$ where the set $R_2$ is obtained from $R_1$ by removing all the sharp signs.

### 4.1.2 Deterministic Case

Let us now turn to the language $\mathcal{L}$. We are interested in the subset of $\mathcal{L}$ denoted above by $\mathcal{M}$ (see Section 4.1).

**Definition 5.** *Let $\Pi(p,t)$ denote the ordered set (the sequence $\{\pi_n(p,t)\}$) of the most general narrowings, written in $\mathcal{L}$, of $t$ with respect to $p$.*[9]

*Let for each $n$ $\varrho_n(p,t)$ be the predicate $(\bigwedge_{i=1}^{n-1} \neg\pi_i(p,t)) \wedge \pi_n(p,t)$ (denoted by $\vec{\pi}_n(p,t)$). The sequence $\{\varrho_n(p,t)\}$ is called a Markov sequence (or ordered set) of the most general narrowings of $t$ with respect to $p$ and denoted by $\overrightarrow{\Pi}(p,t)$. In the sequel we omit the negation part of $\varrho_n(p,t)$ since order allows us to unambiguously restore this part of this predicate.*

The following theorem is a deterministic version of Theorem 1. To deal with this issue we introduce three additional operations $\odot$, $\otimes$, $\check{\otimes}$ as follows.

Given two finite sequences (two ordered sets), the concatenation of the sequences $\{a_1,\ldots,a_n\} \odot \{b_1,\ldots,b_m\}$ is the following sequence $\{c_1,\ldots,c_{n+m}\} = \{a_1,\ldots,a_n,b_1,\ldots,b_m\}$.

Given $v \in \mathcal{E}$, $\langle \pi_1, (v\ \mathtt{:=}\ t_1) \rangle \otimes \langle \overrightarrow{\Pi}, \Theta \rangle (v \mathbin{+\!+} q, g)$ is defined as the ordered set of all elements of $\langle \overrightarrow{\Pi}, \Theta \rangle (v \mathbin{+\!+} q, g)$, in which the pairs $\langle \pi_2, (v\ \mathtt{:=}\ t_2) \rangle$ are replaced by $\langle \pi_1, \lambda \rangle; \langle \pi_2, (v\ \mathtt{:=}\ t_1 \mathbin{+\!+} t_2) \rangle$. I.e. we lengthen the value of the e-variable $v$, and the ordering on this set remains unchanged.

[7]There exists a well-known equation class (named quadratic equations) where, given an equation, each variable occurs at most twice in the equation. The corresponding solution sets are described by a simple algorithm, in terms of finite graphs. See, for example, [4].

[8]The formal normal form of $\pi$, see Section 4.1.

[9]Hence for each $i$ the pattern-matching relation $\pi_i(p,t)$ follows Markov's rule (Section 2.2) choosing at most one witness of the $\pi_i(p,t)$ satisfiability.

If the arguments of the operator $\otimes$ are not of the form above, then this operator coincides with $\oplus$.

The binary operation $\langle\pi_1,\theta_1\rangle \check{\otimes} \langle\overrightarrow{\Pi},\Theta\rangle(p,t)$ results in an ordered set of the narrowings being the formal normal forms of all elements of $\langle\pi_1,\theta_1\rangle \otimes \langle\overrightarrow{\Pi},\Theta\rangle(p,t)$. This operation saves the ordering on this set.

**Theorem 2.** *For any $p \in \mathcal{P}(\mathcal{V})$, $t \in \mathcal{P}({}^{\#}\mathcal{V})$ such that for all $v \in \mathcal{E}(p)$, $\mu_v(p) \le 1$, there exists a finite $\overrightarrow{\Pi}(p,t)$, corresponding to $\Pi(p,t)$, such that for each Markov's narrowing $\phi$ of $t$ w.r.t. $p$ there exists $\pi \in \Pi(p,t)$ such that $t\tilde{\phi} \lll t\tilde{\pi}$ (see Section 2.1).*

*Proof.* The proof below is a simple specialization of the proof of Theorem 1. Throughout this proof one has to interpret the narrowings being considered as written in the language $\mathcal{L}$. That is to say, the pattern matching follows Markov's rule (Section 2.2). First, we specialize the syntax, removing all the asterisk signs from the indices of both the variables and the parameters. We will apply the deterministic semantics of $\mathcal{L}$ to our scenario given above for the nondeterministic case.

**Inductive step:**

The cases **(1)** and **(2)** are almost deterministic. To obtain $\langle\overrightarrow{\Pi},\Theta\rangle(p,t)$ we just need to order the two sequences generating $\langle\Pi,\Theta\rangle(p,t)$ in the subcase **(1.c)**. They may be ordered, for example, according to the second indices of $\pi_{1i}$ (or in the opposite order since the satisfiability sets of the narrowings $\pi_{11}$ and $\pi_{12}$ are disjoint).

The other cases take more effort to be proved. Let $\mathcal{R}$ denote $\langle\overrightarrow{\Pi},\Theta\rangle(p,t)$.

**(3):** *Suppose $p \doteq$ `e.y` ++ $c$ ++ $q$ ++ `e.z`, where $c$ is a character.*

According to Markov's rule for the search of the first occurrence of $c$ in an unknown string we have to order the set $\langle\Pi,\Theta\rangle(p,t)$, excluding the irrelevant narrowings from this set. Thus we scan the parameterized term $t$ from the left to the right and order the narrowings according to the times when they were generated. Recall that such a decomposition of Markov's narrowings is not sound (see Remark 4.1.2), so we have to compose the constructed sequences $\pi$ back to the formal normal form $\check{\pi}$.

**(3.a)** If $t \doteq \lambda$, then $\langle\overrightarrow{\Pi},\Theta\rangle(p,t) = \emptyset$.

**(3.b)** If $t \doteq c$ ++ $g$, then, by the inductive assumption, we can compute $\langle\lambda,(\texttt{e.y := }\lambda)\rangle \oplus \langle\overrightarrow{\Pi},\Theta\rangle(q \texttt{++e.z}, g)$ which may be empty. Hence we have to lengthen the value of `e.y` and take into account the set $\langle\overrightarrow{\Pi},\Theta\rangle(p,g)$. That yields

$\mathcal{R} := (\langle\lambda,(\texttt{e.y := }\lambda)\rangle \oplus \langle\overrightarrow{\Pi},\Theta\rangle(q \texttt{++e.z}, g)) \odot (\langle\lambda,(\texttt{e.y := }c)\rangle \otimes \langle\overrightarrow{\Pi},\Theta\rangle(p,g))$.

By similar arguments one can consider the remaining alternatives as follows.

If $t \doteq c_1$ ++ $g$ where $c \neq c_1 \in \mathcal{C}$, then $\mathcal{R} := \langle\lambda,(\texttt{e.y := }c_1)\rangle \otimes \langle\overrightarrow{\Pi},\Theta\rangle(p,g)$.

If $t \doteq v$ ++ $g$ where $v \in {}^{\#}\mathcal{S}$, then let $\langle\pi,\theta\rangle := \langle(v = c),(\texttt{e.y := }\lambda)\rangle$ and

$\mathcal{R} := (\langle\pi,\theta\rangle \oplus \langle\overrightarrow{\Pi},\Theta\rangle(q \texttt{++e.z}, g\tilde{\pi})) \odot (\langle\lambda,(\texttt{e.y := }v)\rangle \otimes \langle\overrightarrow{\Pi},\Theta\rangle(p,g))$.

Computing the set $\langle\overrightarrow{\Pi},\Theta\rangle(p,g)$ we have to test $g$ by all the variants of the case **(3)**. Since $ln(g) < ln(t)$, the variant **(3.b)** can be involved by the case **(3)** at most finitely many times.

**(3.c)** If $t \doteq \texttt{\#e.w} \texttt{++} g$, then let $\pi$ be the following Markov relation $(\texttt{\#e.w} = \texttt{\#e.w}_{c[1]} \texttt{++} c \texttt{++} \texttt{\#e.r}_{c[1]})$, where $\texttt{\#e.w}_{c[1]}$, $\texttt{\#e.r}_{c[1]}$ are fresh parameters.

By the inductive assumption, we can compute the set $\mathcal{R}_1 = \langle\overrightarrow{\Pi},\Theta\rangle(q \texttt{++e.z}, \texttt{\#e.r}_{c[1]} \texttt{++} g\tilde{\pi})$, but the pure concatenation of two narrowings $\pi$ and $\pi_1$ from $\mathcal{R}_1$ may lead to an incorrect narrowing $\pi;\pi_1$ since, by Markov's rule, the unknown string $\texttt{\#e.w}_{c[1]}$ does not contain the character $c$ and this restriction may contradict to $\pi_1$ and the semantics of $p$ (see Section 4.1

for examples). The formal composition of $\pi$; $\pi_1$ constructs (recovers) the correct narrowing of $t$ w.r.t. $p$, provided that the restriction imposed on the value of $\texttt{\#e.w}_{c[1]}$ is declined and hence the value may be lengthened, if that is required by $p$ and $t$. Thus we make *the only* step of the $\mathcal{L}$-machine being responsible for the search for the value of the variable `e.y`.

$\mathcal{R} := (\langle\pi, (\texttt{e.y := \#e.w}_{c[1]})\rangle \check{\otimes} \mathcal{R}_1) \odot (\langle\lambda, (\texttt{e.y := \#e.w})\rangle \otimes \langle\overrightarrow{\Pi}, \Theta\rangle(p, g))$.

The narrowings from the second argument of $\odot$ are semantics relevant and correspond to lengthening behind the whole unknown string `#e.w`, since `#e.w` is allowed to be without $c$.

The case **(3)** has been proved: the set $\mathcal{R} = \langle\overrightarrow{\Pi}, \Theta\rangle(p, t)$ is finite.

**(4):** *Suppose* $p \doteq \texttt{e.}y \texttt{ ++ } \texttt{s.}x \texttt{ ++ } q \texttt{ ++ } \texttt{e.}z$.

This case follows by arguments similar to the case **(3)**, differing only by the two details mentioned in the proof of Theorem 1. We now consider the last possibility.

**(5):** *Suppose* $p \doteq \texttt{e.}y \texttt{++} v \texttt{++} q$, *where* $v \in \mathcal{E}$.

Since $\mu_{\texttt{e.y}}(p) = \mu_v(p) = 1$ we have $\langle\overrightarrow{\Pi}, \Theta\rangle(p, t) = \langle\lambda, (\texttt{e.y := }\lambda)\rangle \oplus \langle\overrightarrow{\Pi}, \Theta\rangle(v \texttt{ ++ } q, g)$.

The inductive step has been proved. This completes the proof. □

*Remark* 4.1.7. The set $\langle\overrightarrow{\Pi}, \Theta\rangle(p, g)$ generated by the proof above is not minimal. Given $\langle\pi_n, \theta_n\rangle(p, g) \in \langle\overrightarrow{\Pi}, \Theta\rangle(p, g)$ where $n$ is the ordering number, if $\pi_n$ is a tautology, then $\langle\overrightarrow{\Pi}, \Theta\rangle(p, g)$ may be reduced to the following sequence $\langle\pi_1, \theta_1\rangle(p, g), \ldots \langle\pi_n, \theta_n\rangle(p, g)$. More subtle cleaning of the set is a subject for future work.

**The deterministic driving.** Similar to the proof $\Phi_1$ of Theorem 1, the proof of Theorem 2 (denoted by $\Phi_2$) is the major part of the driving algorithm for one-rule programs written in $\mathcal{M} \subset \mathcal{L}$. Let us use denotations analogous to the ones introduced for the description of the nondeterministic driving (see Section 4.1.1). We refer to the proof above as $\Phi_2(\wp)$ where $\wp$ is the set $\langle\overrightarrow{\Pi}, \Theta\rangle$ being transformed by the proof.

Let a program $P = \langle\tau, R\rangle$ written in $\mathcal{M}$ be given, where $\tau = \texttt{f(t}_1\texttt{,...,t}_n\texttt{)}$ and $R$ is $\texttt{f(p}_1\texttt{,...,p}_n\texttt{) = r;}$. The deterministic driving algorithm $\texttt{drv(}\tau\texttt{,'T',}R\texttt{)}$ differs from the non-deterministic one only in the following two details: (1) replace $\Phi_1(\wp)$ by $\Phi_2(\wp)$; (2) enumerate the rules $R_1$ of the resulting program according to the original enumeration of the narrowings from which these rules were produced. By default, the $\mathcal{L}$-semantics bears in mind the removed negation parts of the narrowings.

## 5 Concluding Remarks

Suppose $P = \langle\tau, R\rangle$ is an arbitrary program written in $\mathcal{M}$. Let $\tau = \texttt{f(t}_1\texttt{,...,t}_n\texttt{)}$ and $R$ is a finite sequence of the rules $\texttt{l}_1 \texttt{ = r}_1\texttt{;...l}_k \texttt{ = r}_k\texttt{;}$ where for each $(1 \le i \le k)$, $(\texttt{l}_i \texttt{ = r}_i\texttt{;})$ is $\texttt{f(p}_{1i}\texttt{,...,p}_{ni}\texttt{) = r}_i\texttt{;}$ (denoted by $R_i$) and for each $v \in \mathcal{E}(\texttt{l}_i)$, $\mu_v(\texttt{l}_i) = 1$. Let $\texttt{g}_1\texttt{,...,g}_m$ be a certain enumeration over the set ${}^{\#}\mathcal{V}(\tau)$. The driving algorithm $\texttt{drv(}\tau\texttt{,'T',}R\texttt{)}$ may work as follows: (1) For each $(1 \le i \le k)$, $Q_i := \texttt{drv(}\tau\texttt{,'T',}R_i\texttt{)}$; where each rule from $Q_i$ is of the form $\texttt{f}_2\texttt{(q}_{(1,ij)}\texttt{, ..., q}_{(m,ij)}\texttt{) = r}_{ij}\texttt{;}$ (2) Let $Q$ denote the sequence obtained by concatenation of all the sequences $Q_i$ according to their numbering. Let $\tau_2$ be $\texttt{f}_2\texttt{(g}_1\texttt{, ..., g}_m\texttt{)}$. The algorithm $\texttt{drv(}\tau\texttt{,'T',}R\texttt{)}$ results in the program $P_2 = \langle\tau_2, Q\rangle$.

In this paper we have suggested two algorithms for one-step unfolding of the programs belonging to the languages $\mathcal{M}$ and $\mathcal{M}_*$. The deterministic programming language $\mathcal{M}$ is Turing-complete. Given a word equation, the algorithm suggested for one-step unfolding of the programs written in the nondeterministic language $\mathcal{M}_*$ results in a finite description of the solution set of the equation.

As far as we know there exist no papers using the nondeterministic pattern-matching as a language for such a description and none considered the class of the equations characterized by Theorems 1, 2 above, namely the variables separated by the two equation sides such that at least one of the sides has at most one occurrence of each word variable.

The author is grateful to Alexei Lisitsa and Antonina Nepejvoda for friendly and very fruitful discussions inspired him to write this article.

## References

[1] N. Bjørner, N. Tillmann, and A. Voronkov. Path feasibility analysis for string-manipulating programs. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2009)*, volume 5505 of *LNCS*, pages 307–321. Springer Berlin Heidelberg, 2009.

[2] A. S. Christensen, A. Møller, and M. I. Schwartzbach. Precise analysis of string expressions. In *10th Int. Static Analysis Symposium (SAS 2003)*, volume 2694 of *LNCS*, pages 1–18, 2003.

[3] A. Colmerauer. An introduction to Prolog III. *Communications of the ACM*, 33(7):69–90, 1990.

[4] V. Diekert. Makanin's algorithm, **chapter 12**. In M. Lothaire, editor, *Algebraic Combinatorics on Words*, pages 387–442. Cambridge University Press, 2002.

[5] M. Emmi, R. Majumdar, and K. Sen. Dynamic test input generation for database applications. In *the Proc. of ISSTA'07*, pages 151–162. ACM, 2007.

[6] X. Fu, X. Lu, B. Peltsverger, S. Chen, K. Qian, , and L. Tao. A static analysis framework for detecting SQL injection vulnerabilities. In *COMPSAC'07*, pages 87–94. IEEE, July 2007.

[7] M. Gabbrielli, M. Ch. Meo, P. Tacchella, and H. Wiklicky. Unfolding for CHR programs. *Theory and Practice of Logic Programming*, FirstView:1–48, October 2013.

[8] P. Godefroid, A. Kiezun, and M. Y. Levin. Grammar-based whitebox fuzzing. In *PLDI'08*, pages 206–215. ACM, 2008.

[9] J. Jaffar. Minimal and complete word unification. *Journal of the ACM*, 37(1):47–85, Jan. 1990.

[10] Neil D. Jones. The essence of program transformation by partial evaluation and driving. In M. Sato, N. D. Jones, and M. Hagiya, editors, *Logic, Language and Computation, a Festschrift in honor of Satoru Takasu*, pages 206–224. Springer-Verlag, April 1994.

[11] A. P. Lisitsa and A. P. Nemytykh. Reachability analysis in verification via supercompilation. *International Journal of Foundations of Computer Science*, 19(4):953–970, August 2008.

[12] G. S. Makanin. The problem of solvability of equations in a free semigroup. (in Russian). *Matematicheskii Sbornik*, 103(2):147–236, 1977. Translation in: Math. USSR-Sb., 32, pp: 129–198, 1977.

[13] A. A. Markov. The theory of algorithms. *AMS Translations*, 2(15):1–14, 1960.

[14] A. Pettorossi, M. Proietti, and S. Renault. Enhancing partial deduction via unfold/fold rules. In J. Gallagher, editor, *Proceedings of Logic Program Synthesis and Transformation, 6th International Workshop*, volume 1207 of *LNCS*, pages 147–168. IEEE Computer Society, 1997.

[15] H. Ruan, J. Zhang, and J. Yan. Test data generation for c programs with string-handling functions. In *TASE'08*, pages 219–226. IEEE, 2008.

[16] D. Shannon, S. Hajra, A. Lee, D. Zhan, and S. Khurshid. Abstracting symbolic execution with string analysis. In *Testing: Academic and Industrial Conference Practice and Research Techniques-MUTATION*, pages 13–22. IEEE, Sept. 2007.

[17] M. H. Sørensen, R. Glück, and N. D. Jones. A positive supercompiler. *Journal of Functional Programming*, 6(6):811–838, 1996.

[18] V. F. Turchin. The language Refal – the theory of compilation and metasystem analysis. Technical Report 20, Courant Institute, New York University, February 1980.

[19] V. F. Turchin. The concept of a supercompiler. *ACM Transactions on Programming Languages and Systems*, 8(3):292–325, 1986.

[20] V. F. Turchin. *Refal-5, Programming Guide and Reference Manual.* New England Publishing Co., Holyoke, Massachusetts, 1989. E-version: `http://www.botik.ru/pub/local/scp/refal5/`, 2000.

[21] V. F. Turchin. Program transformation with metasystem transitions. *Journal of Functional Programming*, 3(3):283–313, 1993.

[22] V. F. Turchin, D. V. Turchin, A. P. Konyshev, and A. P. Nemytykh. Refal-5: Sources, executable modules. [online], 2000. `http://www.botik.ru/pub/local/scp/refal5/`.

[23] G. Wassermann and Z. Su. Sound and precise analysis of web applications for injection vulnerabilities. In *PLDI'2007*, pages 32–41. Springer-Verlag, June 2007.

## A The Proof of the Case (4), Theorem 1

**(4):** *Suppose* $p \doteq \mathtt{e.y}_* \mathbin{{+}{+}} \mathtt{s.x}_* \mathbin{{+}{+}} q \mathbin{{+}{+}} \mathtt{e.z}_*$. Let $\tau$ denote $c \in \mathcal{C}$ or $v \in {}^{\#}\mathcal{S}$.

Let us turn to the concrete details.

If $\mu_\tau(t) > 0$, then there exist $t_{\tau[i]}, g_{\tau[i]} \in \mathcal{P}({}^{\#}V)$ such that $t \doteq t_{\tau[i]} \mathbin{{+}{+}} \tau_{[i]} \mathbin{{+}{+}} g_{\tau[i]}$, where we use the denotations introduced in the previous case. We can compute the following set of the narrowings $B = \langle \Pi, \Theta \rangle (q \mathbin{{+}{+}} \mathtt{e.z}_*, g_{\tau[i]})$.

Suppose $\mu_{\mathtt{s.x}_*}(p) > 1$, then each $\langle \pi, \theta \rangle (q \mathbin{{+}{+}} \mathtt{e.z}_*, g_{\tau[i]})$ includes a pair of the kind $\langle \rho, (\mathtt{s.x}_* := \tau_2) \rangle$, where $\tau_2$ is a character $c_2$ or $v_2 \in {}^{\#}\mathcal{S}$.

Let $\pi_\tau$ denote the relation $(\tau = \tau_2)$, which may be a contradiction, a tautology (denoted by $\lambda$), either $(\mathtt{\#s.u}_* = \mathtt{\#s.u}_{*2})$ or $(\mathtt{\#s.u}_* = c_2)$ or $(\mathtt{\#s.u}_{*2} = c)$. Here the numbered terms are the instances of $\tau_2$, while the non-numbered terms are the instances of $\tau$.

Let $B|_{(\mathtt{s.x}_* := \tau)}$ denote the subset of the matching substitutions from $B$ compatible with $(\mathtt{s.x}_* := \tau)$. $(\langle \pi_\tau, (\mathtt{e.y}_* := t_{\tau[i]} \tilde{\pi}_\tau) \rangle \oplus \langle \Pi, \Theta \rangle (q \mathbin{{+}{+}} \mathtt{e.z}_*, g_{\tau[i]} \tilde{\pi}_\tau)|_{(\mathtt{s.x}_* := \tau)}) \subset \langle \Pi, \Theta \rangle (p, t)$.[10]

If $\mu_{\mathtt{s.x}_*}(p) = 1$, then $(\langle \lambda, (\mathtt{e.y}_* := t_{\tau[i]}) \rangle; \langle \lambda, (\mathtt{s.x}_* := \tau) \rangle \oplus \langle \Pi, \Theta \rangle (q \mathbin{{+}{+}} \mathtt{e.z}_*, g_{\tau[i]})) \subset \langle \Pi, \Theta \rangle (p, t)$.

If $\tau$ is $\mathtt{\#e.w}_*$ ($\tau \in {}^{\#}\mathcal{E}$), $\mu_\tau(t) > 0$ and the unknown string $\mathtt{\#e.w}_*$ is not $\lambda$, then it includes an unknown character. Let $\mathtt{\#e.w}_{(\tau[i],\mathtt{\#s.u}[*])}$, $\mathtt{\#s.u}_{(\tau[i],[*])}$, $\mathtt{\#e.r}_{(\tau[i],\mathtt{\#s.u}[*])}$ be fresh parameters, then $\mathtt{\#e.w}_* = \mathtt{\#e.w}_{(\tau[i],\mathtt{\#s.u}[*])} \mathbin{{+}{+}} \mathtt{\#s.u}_{(\tau[i],[*])} \mathbin{{+}{+}} \mathtt{\#e.r}_{(\tau[i],\mathtt{\#s.u}[*])}$ (denoted by $\kappa_{(\tau,\mathtt{\#s.u}_{(\tau[i],[*])})}$) meaning that the unknown character $\mathtt{\#s.u}_{(\tau[i],[*])}$ may take any position in the unknown string $\mathtt{\#e.w}_*$. For each $(0 < i \leq \mu_\tau(t))$ the set

$$\langle \kappa_{(\tau,\mathtt{\#s.u}_{(\tau[i],[*])})}, (\mathtt{e.y}_* := t_{\tau[i]} \tilde{\kappa}_{(\tau,\mathtt{\#s.u}_{(\tau[i],[*])})} \mathbin{{+}{+}} \mathtt{\#e.w}_{(\tau[i],\mathtt{\#s.u}[*])}) \rangle;\ \langle \lambda, (\mathtt{s.x}_* := \mathtt{\#s.u}_{(\tau[i],[*])}) \rangle$$
$$\oplus \langle \Pi, \Theta \rangle (q \mathbin{{+}{+}} \mathtt{e.z}_*, \mathtt{\#e.r}_{(\tau[i],\mathtt{\#s.u}[*])} \mathbin{{+}{+}} g_{\mathtt{\#s.u}_*[i]} \tilde{\kappa}_{(\tau,\mathtt{\#s.u}_{(\tau[i],[*])})})$$

is a subset of $\langle \Pi, \Theta \rangle (p, t)$.

We now compute the set $\langle \Pi, \Theta \rangle (p, t)$. If $\mu_{\mathtt{s.x}_*}(p) = 1$, then it is

$$\bigcup_{\tau \in \{b | (b \in \mathcal{C}) \wedge (\mu_b(t) > 0)\} \cup {}^{\#}\mathcal{S}(t)} (\bigcup_{1 \leq i \leq \mu_\tau(t)} ( \langle \lambda, (\mathtt{e.y}_* := t_{\tau[i]}) \rangle; \langle \lambda, (\mathtt{s.x}_* := \tau) \rangle$$
$$\oplus \langle \Pi, \Theta \rangle (q \mathbin{{+}{+}} \mathtt{e.z}_*, g_{\tau[i]}))) \cup$$
$$\bigcup_{\tau \in {}^{\#}\mathcal{E}(t)} (\bigcup_{1 \leq i \leq \mu_\tau(t)} ( \langle \kappa_{(\tau,\mathtt{\#s.u}_{(\tau[i],[*])})}, (\mathtt{e.y}_* := t_{\tau[i]} \tilde{\kappa}_{(\tau,\mathtt{\#s.u}_{(\tau[i],[*])})} \mathbin{{+}{+}} \mathtt{\#e.w}_{(\tau[i],\mathtt{\#s.u}[*])}) \rangle;$$
$$\langle \lambda, (\mathtt{s.x}_* := \mathtt{\#s.u}_{(\tau[i],[*])}) \rangle$$
$$\oplus \langle \Pi, \Theta \rangle (q \mathbin{{+}{+}} \mathtt{e.z}_*, \mathtt{\#e.r}_{(\tau[i],\mathtt{\#s.u}[*])} \mathbin{{+}{+}} g_{\mathtt{\#s.u}_*[i]} \tilde{\kappa}_{(\tau,\mathtt{\#s.u}_{(\tau[i],[*])})})))$$

If $\mu_{\mathtt{s.x}}(p) > 1$, then it is

$$\bigcup_{\tau \in \{b | (b \in \mathcal{C}) \wedge (\mu_b(t) > 0)\} \cup {}^{\#}\mathcal{S}(t)} (\bigcup_{1 \leq i \leq \mu_\tau(t)} ( \langle \pi_\tau, (\mathtt{e.y}_* := t_{\tau[i]} \tilde{\pi}_\tau) \rangle$$
$$\oplus \langle \Pi, \Theta \rangle (q \mathbin{{+}{+}} \mathtt{e.z}_*, g_{\tau[i]} \tilde{\pi}_\tau)|_{(\mathtt{s.x}_* := \tau)}) \cup$$

[10] Notice that we do not insert the corresponding new assignment to $\mathtt{s.x}_*$. The old assignment to $\mathtt{s.x}_*$ generated by the inductive assumption is compatible with the new one. Hence for each variable $v \in \mathcal{V}(p)$ and for each narrowing $\delta$ from $\langle \Pi, \Theta \rangle (p, t)$ there is at most one assignment to $v$ in $\delta$.

$$\begin{array}{ll}\bigcup_{\tau\in{}^{\#}\mathcal{E}(t)}(\bigcup_{1\leq i\leq\mu_\tau(t)}( & \langle\pi_\tau,\lambda\rangle; \\ & \langle\kappa_{(\tau,\#\mathtt{s.u}_{(\tau[i],[*])})},(\mathtt{e.y}_* := t_{\tau[i]}\tilde{\kappa}_{(\tau,\#\mathtt{s.u}_{(\tau[i],[*])})}\;\mathtt{++}\;\#\mathtt{e.w}_{(\tau[i],\#\mathtt{s.u}[*])}))\rangle \\ & \oplus\ \langle\Pi,\Theta\rangle(q\;\mathtt{++}\;\mathtt{e.z}_*,\#\mathtt{e.r}_{(\tau[i],\#\mathtt{s.u}[*])}\;\mathtt{++}\;g_{\#\mathtt{s.u}[i]}\tilde{\kappa}_{(\tau,\#\mathtt{s.u}_{(\tau[i],[*])})})))\end{array}$$

Here $\pi_\tau$ was defined above and, because of the parameters' freshness, in the union over $\tau\in{}^{\#}\mathcal{E}(t)$ all the substitutions created by the inductive assumption are unconditionally compatible with $(\mathtt{s.x}_* := \tau)$. The case **(4)** has been proved: the set $\langle\Pi,\Theta\rangle(p,t)$ is finite.