Joint NCC & IIS Bull., Comp. Science, 8 (1998), 1–18
© 1998 NCC Publisher

A method for simplification procedure design based on formula rewriting systems*

I. S. Anureev

We generalize the notion of formula rewriting systems proposed in the preliminary work, extend the class of such systems and expand our previous results for the new class. We demonstrate that the well-known techniques of formula simplification can be represented in terms of this formalism and investigate termination problem for such systems in detail.

Introduction

A powerful method for design of simplification procedures preserving satisfiability is proposed. The need for such method can be motivated by the following reasons: (i) There are important applications of simplification procedures preserving satisfiability; (ii) Known methods of simplification procedures design do not cover all such applications.

Let us consider two applications of simplification procedures preserving satisfiability (constraint-based deduction and program verification) and two well-known methods of simplification procedures design (term rewriting systems and narrowing).

Constraint-based deduction. In constraint-based deduction it is often sufficient to check constraints for satisfiability and postpone their solving until a later stage [1]. Formula rewriting is one of the methods of checking constraints for satisfiability [2]. A particular set of formulas (called solved forms) and associated set of elementary constraint solvers (checking any solved forms for satisfiability) are distinguished. Then a set of terminating rewriting rules on formulas preserving satisfiability, is designed. If every irreducible formula is a solved form, then a method of reducing a wider class of formulas to the set of solved forms is obtained. Therefore, formula rewriting is a method of design of constraint simplifiers preserving satisfiability, and development of formula rewriting formalisms is an important problem.

Program verification. In the classical program verification, the problem

*Partially supported by INTAS–RFBR under Grant 95–0378.

of program correctness is reduced to proving validity of several formulas (verification conditions). Since a formula is valid iff its negation is not satisfiable, validity problem can be reduced to satisfiability problem and design of simplification procedures preserving satisfiability plays an important role in this case, too.

Term rewriting systems. Nowadays the use of term rewriting systems is the most wide-spread method of designing simplification procedures. However, the replacement 'equal by equal' (underlying term rewriting systems) is inhibitory for the use of more powerful simplifications. In particular, we cannot apply such simplifications to case analysis and replacement of variables since they are not equivalent formula transformations.

Narrowing. A well-known technique to solve equations in equational theories is the 'narrowing' transformation on terms. Narrowing can be also considered as a method of designing simplification procedures. The advantage of this method, in particular, is simulation of replacement of variables. Unfortunately, in the general case narrowing does not preserve satisfiability.

The technique of formula rewriting systems ([3, 4]) can simulate both formula rewriting technique (based on term rewriting systems) and narrowing as well as replacement of variables and case analysis. The other important feature of formula rewriting systems is that the sufficient conditions of preserving satisfiability for such systems are related to algebraic structures. This allows us to design problem-oriented simplification procedures.

A structure of the paper. In Section 1 we remind the well-known logical and set-theoretic notions. Generalized definitions of formula rewriting system and reduction relation generated by it are presented in Section 2. In the same section we demonstrate that the termination property is undecidable for formula rewriting systems. In Section 3 we explain the way the well-known techniques of formula simplification can be embedded into formalism of formula rewriting systems. Sufficient conditions of preserving satisfiability for formula rewriting systems are stated in Section 4. The rest of the paper is devoted to termination property for formula rewriting systems. Relation between the termination of formula rewriting systems and simplification ordering is studied in Section 5. A termination of formula rewriting systems with respect to reduction strategies is considered in Sections 6 and 7. A special class of formula rewriting systems (selector elimination systems) is introduced in Section 6. In Section 7 selector elimination systems are generalized to argument status. In Section 8 the main introduced concepts are illustrated by examples from the tuple theory. In the conclusion further research is discussed.

1. Preliminaries

1.1. Terms, formulas and substitutions

The reader may refer to [5] for the concepts of terms, substitutions and unifiers. The notations used in this paper are listed below.

For a set S, multiset U and $u \in U$, $|S|$ denotes the cardinality of S, $\mathcal{FS}(S)$ denotes the set of all finite subsets of S, $FM(S)$ denotes the set of all finite multisets of elements of S and $\mathcal{O}(u, U)$ denotes the number of occurrences u in U.

For a signature $\Sigma = (\mathcal{F}, \mathcal{P}, \mathcal{V})$ with the set of function symbols $\mathcal{F}$, the set of predicate symbols $\mathcal{P}$ and the set of variables $\mathcal{V}$, $\mathcal{T}$ denotes the set of Σ-terms, $\mathcal{UF}$ denotes the set of unquantified Σ-formulas with equality $=$, $\mathcal{E} = \mathcal{T} \cup \mathcal{UF}$ denotes the set of Σ-expressions and $\mathcal{S}$ denotes the set of substitutions over $\mathcal{E}$.

For $u \in \mathcal{E}$ and $\sigma \in \mathcal{S}$, $\mathcal{V}ar(u)$ denotes the set of variables of u, $\mathcal{MV}ar(u)$ denotes the multiset of variables (in respect of their occurrences) of u, $root(u)$ denotes the root of u, $|u|$ denotes the number of occurrences of functional and predicate symbols into u, $\mathcal{P}(u)$ denotes the set of positions of u with Λ as the top most position, $\mathcal{D}om(\sigma)$ denotes the domain of σ and $\mathcal{VR}ange(\sigma)$ denotes the variable range of σ.

For distinct variables $x_1, \ldots, x_n$ and $t_1, \ldots, t_n \in \mathcal{T}$,

$$(x_1 \to t_1, \ldots, x_n \to t_n)$$

denotes the substitution σ such that $\mathcal{D}om(\sigma) \subseteq \{x_1, \ldots, x_n\}$ and $x_i\sigma = t_i$ for each $1 \leq i \leq n$. In particular, ( ) is an identity substitution.

A renaming substitution is a bijective substitution. This implies that a renaming τ, restricted to the set of variables $X = \mathcal{D}om(\tau)$, is a permutation of X. Note that the composition $\sigma\tau$ of renamings σ, τ is also a renaming, and that the inverse τ^{-1} of a renaming τ exists and it is also a renaming. We call θ a unifier of substitutions σ and σ' if $\sigma\theta = \sigma'\theta$. It is a most general unifier (mgu) of σ and σ' if for every uinfier θ' of σ and σ' there exists a renaming τ such that $\theta' = \theta\tau$. Any two substitutions which are unifiable (have a unifier) have a mgu. It is unique modulo renamings.

Problem of finding a mgu of substitutions σ and σ' can be viewed as the problem of solving the set of equations $\{x\sigma = x\sigma' | x \in \mathcal{D}om(\sigma) \cup \mathcal{D}om(\sigma')\}$. Let E be a set of equations and let $t, t_1, \ldots, t_n, s_1, \ldots, s_n \in \mathcal{T}$, $f \in \mathcal{F}$ and $x \in \mathcal{V}$. The algoritm exploiting this representation consists of the following rules which transform one set of equations into another one:

1. Term decomposition:

$$\{f(t_1, \ldots, t_n) = f(s_1, \ldots, s_n)\} \cup E \to_{(\,)} \{t_1 = s_1, \ldots, t_n = s_n\} \cup E;$$

2. Removal of trivial equations: $\{t = t\} \cup E \rightarrow_{(\,)} E$;
3. Swap: $\{t = x\} \cup E \rightarrow_{(\,)} \{x = t\} \cup E$ if t is not a variable;
4. Variable elimination: $\{x = t\} \cup E \rightarrow_{(x \rightarrow t)} E(x \rightarrow t)$ if $x \notin \mathcal{V}ar(t)$. For $E = \{e_1, \ldots, e_n\}$, $E\sigma = \{e_1\sigma, \ldots, e_n\sigma\}$.

A finite derivation sequence $\xi = E_0 \rightarrow_{\sigma_1} E_1 \rightarrow_{\sigma_2} \ldots \rightarrow_{\sigma_2} E_n$ is successful if none of the above rules is applied to E_n. In this case, the substitution $\sigma_1\sigma_2 \ldots \sigma_n$ is called a computed answer substitution of the successful derivation ξ. We have the following well-known theorem:

Theorem. *Let $E = \{x\sigma = x\sigma' | x \in \mathcal{D}om(\sigma) \cup \mathcal{D}om(\sigma')\}$. Then the following statements are equivalent: (i) The substitutions σ and σ' are unifiable; (ii) There is a* mgu *of the substitutions σ and σ'; (iii) The derivation tree with the root E and constructed with the above rules is finite and has only successful branches, all yielding a* mgu *of the substitutions σ and σ as the computed answer substitution. Furthermore, if the substitutions σ and σ' are not unifiable, the derivation tree is also finite, but now with all branches ending unsuccessfully.*

1.2. Relations and orderings

Let us remind some properties of relations and orderings. Let S be a set and $\rightarrow$ be a binary relation on S. As usual, $\rightarrow^+$ denotes the transitive closure of $\rightarrow$.

Definition. A relation $\rightarrow$ is terminating if there are no infinite chains $s_1 \rightarrow s_2 \rightarrow \ldots$ of elements of S.

Let $u, v \in S$ and $U, W \in FM(S)$. For a binary relation $\rightarrow$ on S, $\rightarrow^m$ denotes the relation $\{(\{u\} \cup W, \{v\} \cup W) | u \rightarrow v\}^+$. For a relation $\rightarrow$ on $S \times FM(S)$, $\rightarrow^m$ denotes the relation $\{(\{u\} \cup W, U \cup W) | u \rightarrow U\}^+$.

Let $u, v \in \mathcal{UF}$, $U, V \in FM(\mathcal{UF})$ and τ be a renaming. Then $\doteq$ denotes the minimal relation such that (i) $u \doteq u\tau$ for any u and τ; (ii) $\emptyset \doteq \emptyset$; (iii) for any U, V, u and v, $U \cup \{u\} \doteq V \cup \{v\}$ if $U \doteq V$ and $u \doteq v$.

For $u, v, u', v' \in \mathcal{UF}$, the relations $\rightarrow_a$ and $\rightarrow_b$, $u \rightarrow_a v$ and $u' \rightarrow_b v'$, $\dot{\subseteq}$ denotes the relation $\{(\rightarrow_a, \rightarrow_b) | \forall u \forall v \exists u' \exists v' \ (u \doteq u'$ and $v \doteq v')\}$ and $\doteq$ denotes the relation $\{(\rightarrow_a, \rightarrow_b) | \rightarrow_a \dot{\subseteq} \rightarrow_b$ and $\rightarrow_b \dot{\subseteq} \rightarrow_a\}$. Let $\doteq^m$ denote the relation $\{(\rightarrow_a, \rightarrow_b) | \rightarrow_a^m \doteq \rightarrow_b^m\}$.

Definition. Let $\succ$ be a strict partial ordering on S. $\succ$ is well-founded if there are no infinite descending chains $s_1 \succ s_2 \succ \ldots$ of elements of S.

Let $M, M', Y \in FM(S)$, $\emptyset \neq X \subseteq M$ and $M' = (M \setminus X) \cup Y$ and let $x \in X$ and $y \in Y$. A relation $\succ_m$ is a multiset extension of $\succ$ if $\succ_m = \{(M, M') | \exists X \exists Y \forall y \exists x (x \succ y)\}$.

Definition. Let $\succ$ be a strict partial ordering on $\mathcal{T}$, $t, s \in \mathcal{T}$, $f \in \mathcal{F}$ and $\sigma \in \mathcal{S}$. So, $\succ$ is monotonic if $\forall s \forall t \forall f\ (s \succ t \Rightarrow f(\ldots, s, \ldots) \succ f(\ldots, t, \ldots))$; $\succ$ is a reduction ordering if $\succ$ is monotonic and well-founded; $\succ$ is stable w.r.t. substitutions if $\forall s \forall t \forall \sigma\ (t \succ s \Rightarrow t\sigma \succ s\sigma)$; $\succ$ is a term ordering if $\succ$ is reduction ordering stable w.r.t. substitutions; $\succ$ possesses the subterm property if $\forall t \forall f\ (s \succ t \Rightarrow f(\ldots, t, \ldots) \succ t)$; $\succ$ is a simplification ordering if $\succ$ is monotonic and $\succ$ possesses the subterm property.

Hereinafter, the set of natural numbers is denoted by $\mathcal{N}$ and $>$ denotes the natural ordering on $\mathcal{N}$.

2. Formula rewriting systems

2.1. Definitions

Throughout the paper, we use the letters p, A and B to represent unquantified formulas, the letters u, u', v, l, r and s to represent expressions, the letters x, y and z to represent variables and the letters σ, σ', θ, ϕ, μ and τ to represent substitutions.

Definition. A triple $p|l \to r$ is a conditional term rewriting rule if l and r are both either terms or formulas. A finite set of conditional term rewriting rules is called a conditional term rewriting system (CTRS).

Let us note that there are no usual restrictions [6] of the form $l \notin \mathcal{V}$ or $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$ in this definition. Throughout the paper, $\mathcal{B}$ denotes some CTRS, $\pi \in \mathcal{B}$, $\pi = p|l \to r$ and $\rho = (\mathcal{B}, s)$.

Definition. A pair ρ is a formula rewriting rule if $\forall \pi$ (l and s are unifiable). Here $\mathcal{B}$ is called a base of ρ, π is called a branch of ρ and s is called a sample of ρ. A set of formula rewriting rules is called a formula rewriting system (FRS).

Definition. Let θ be a mgu of σ and σ', and let $\bar{x} = \mathcal{D}om(\sigma) \cup \mathcal{D}om(\sigma')$, $\bar{y} = \mathcal{V}ar(\bar{x}\sigma = \bar{x}\sigma')$ and $\bar{z} = \mathcal{VR}ange(\theta)$. θ is a trivial unifier of σ and σ' if the formula $\forall \bar{y} \exists \bar{z} (\bar{x}\sigma = \bar{x}\sigma' \Rightarrow \bar{y} = \bar{y}\theta)$ is valid. Substitutions σ and σ' are trivially unifiable if they have a trivial unifier.

As an obvious consequence of the definition we have

Proposition 1. *Two substitutions are trivially unifiable iff a trivial unifier of these substitutions can be found by the unification algorithm without the term reduction rule.*

Throughout the paper, let $q \in \mathcal{P}(A)$ and $l\sigma = A_q$, let θ be a mgu of l and s, ϕ be a trivial unifier of σ and θ, and R be a FRS, and let $\rho \in R$. To avoid variable confusion, the following restrictions hold: A and R do not contain common variables,

$$\begin{aligned}&\mathcal{D}om(\theta) = \mathcal{V}ar(l) \cup \mathcal{V}ar(s),\\ &\mathcal{D}om(\phi) = \mathcal{V}ar(\{x\sigma = x\theta | x \in \mathcal{D}om(\sigma) \cup \mathcal{D}om(\theta)\})\end{aligned}$$

and the sets $\mathcal{VR}ange(\theta)$ and $\mathcal{VR}ange(\phi)$ consist of new variables.

Definition. A relation $\rightarrow_R$ is a reduction relation generated by R if

$$\rightarrow_R = \{(A, \{(p\sigma \wedge A[r\sigma]_q)\phi | \pi\}) | A, q, \rho\}.$$

Let $W \in FM(\mathcal{UF})$ and $w \in W$. A relation $(\rightarrow_R)_b$ is a branch relation generated by R if

$$(\rightarrow_R)_b = \{(A, w) | \exists W (A \rightarrow_R W)\}.$$

2.2. Properties

Let $\mathcal{A} = (FM(\mathcal{UF}), (\rightarrow_R)^m)$. For each R there is a corresponding abstract reduction system [6], namely $\mathcal{A}$. Therefore all notions (termination, confluence, normal form and so on) and their properties defined in the theory of abstract reduction systems are carried over to FRSs via the associated abstract reduction system. For instance, R is terminating if $\mathcal{A}$ is terminating. Let us examine thoroughly the termination property which is of a considerable importance in the simplification procedures design. Unfortunately, the termination property is in general undecidable for formula rewriting systems.

Theorem 1. *The termination property is undecidable for formula rewriting systems.*

Proof. Suppose, for a proof by contradiction, that the termination property is decidable for formula rewriting systems. Let $R_t = \{l \rightarrow r\}$ be a TRS. Let $s = l$, $\pi = l \rightarrow r$, $\rho = (\{\pi\}, s)$, $R = \{\rho\}$ and $\mathcal{FP} = \mathcal{UF} \times \mathcal{UF}$. We have the following chain of easily proved conjectures: (i) R_t is terminating iff $\rightarrow \ = ((\rightarrow_{R_t})_{|\mathcal{FP}})^+$ is terminating; (ii) $\rightarrow$ is terminating iff $((\rightarrow_R)_b)^+$ is terminating; (iii) $((\rightarrow_R)_b)^+$ is terminating iff R is terminating. By the conjectures, R is terminating iff R_t is terminating. However, this is in contradiction with the fact the termination property is undecidable for term rewriting systems with only one rule [7]. $\square$

There are two ways to design terminating simplification procedures based on formula rewriting systems: (i) to distinguish some classes of terminating formula rewriting systems; (ii) to impose restrictions (strategies) on the order of formula rewriting. Both ways will be considered in the next sections. Here, we only give definitions of several strategies.

Definition. A relation $\rightarrow \subseteq \rightarrow_R$ is called a strategy for R. Let $\rightarrow_{Rs}$ be a strategy for R, $q = \Lambda$, $\sigma = ()$. Thus, $\rightarrow_{Rs}$ is an apply-to-sample strategy if

$$\rightarrow_{Rs} = \{(A, \{(p\sigma \wedge A[r\sigma]_q)\phi|\pi\})|A, q, \rho\},$$

and u is a redex of R if

$$\exists\rho\exists\sigma\exists q\exists\theta(u = s\sigma \text{ and } \forall\pi(\sigma \text{ and } \theta \text{ are trivially unified})).$$

Let $\rightarrow_{Ri}$ be a strategy for R and u be a proper subexpression of A_q and let $\forall u(u$ is not redex of $R)$. The relation $\rightarrow_{Ri}$ is an any-innermost strategy if

$$\rightarrow_{Ri} = \{(A, \{(p\sigma \wedge A[r\sigma]_q)\phi|\pi\})|A, q, \rho\}.$$

3. Representation of simplifying transformations by formula rewriting systems

Let us demonstrate how well-known techniques of formula simplification can be embedded into formalism of formula rewriting systems.

Term rewriting systems. A pair $l \rightarrow r$ is a term rewriting rule if l is not a variable and $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$. A set of term rewriting rules is called a term rewriting system (TRS). Let R_t be a TRS and let $\rho_t = l \rightarrow r \in R_t$ and $A_q = l\sigma$. A relation $\rightarrow_{R_t}$ is a reduction relation generated by R_t if $\rightarrow_{R_t} = \{(A, A[r\sigma]_q)|A, q, \rho_t\}$.

Proposition 2. *Let $R = \{(\{l \rightarrow r\}, l)|\rho\}$ and $\mathcal{FP} = \mathcal{UF} \times \mathcal{UF}$. Then* $(\rightarrow_{R_t})_{|\mathcal{FP}} \overset{m}{\doteq} \rightarrow_R$.

Narrowing. Let $\rightsquigarrow_{R_t}$ be a binary relation on $\mathcal{UF}$ and let $A_q \notin \mathcal{V}$. A relation $\rightsquigarrow_{R_t}$ is a narrowing relation if $\rightsquigarrow_{R_t} = \{(A, A[r]_q\theta)|q, A, \rho_t\}$. Let $u, v, w \in \mathcal{E}$. An expression u is an example of v if $\exists\sigma(u = v\sigma)$.

Let $\mathcal{E}x(v) = \{u | u \text{ is an example of } v\}$ and let $U \subseteq \mathcal{E}$, $u, v \in U$, $\mu \in \mathcal{S}$, $u = v\mu$ and $x \in \mathcal{V}$. A set U covers w if $\forall u \forall v \exists x\ (root(x\mu) \notin \mathcal{V})$ and $\forall u(w \in \mathcal{E}x(u) \text{ and } u \notin \mathcal{V})$.

Example. Let $U = \{f(z, z'), f(g(z), z'), f(z, h(z')), f(g(z), h(z'))\}$ and $w = f(g(x), h(y))$. Then U covers w.

Proposition 3. *Let U cover l, $s \in U$ and $R = \{(\{l \to r\}, s) | s, \rho_t\}$. Then* $\rightsquigarrow_{R_t} \overset{\cdot}{=}^{m} \to_R$.

Case analysis. A relation on $\mathcal{UF} \times \mathcal{FS}(\mathcal{UF})$ is called a case analysis.

Proposition 4. *Let $\to$ be a case analysis and let $A \to U$, $u \in U$ and $R = \{(\{A \to u | u\}, A) | A, U\}$. Then $\to \overset{\cdot}{=}^{m} \to_{Rs}$.*

Replacement of variables. Let $\to$ be a binary relation on $\mathcal{UF}$ and let $\mathcal{D}om(\sigma) = \mathcal{V}ar(A)$, $\mathcal{D}om(\sigma) \cap \mathcal{VR}ange(\sigma) = \emptyset$ and $B \in \mathcal{UF}$. A relation $\to$ is a replacement of variables if $\forall A \forall B (A \to B \Rightarrow \exists \sigma (B = A\sigma))$.

Proposition 5. *Let $\to$ be a replacement of variables and let $A \to B$ and $R = \{(\{B \to B\}, A) | A, B\}$. Then $\to \overset{\cdot}{=}^{m} \to_{Rs}$.*

The proof of the above propositions is straightforward from definitions.

4. Preserving satisfiability

Let us propose sufficient conditions of preserving satisfiability for formula rewriting systems. Throughout the rest of the paper, we consider only finite formula rewriting systems (finite sets of rules) and use the letter $\mathcal{A}$ to denote algebraic structures. A validity and a satisfiability will be considered to mean a validity and a satisfiability in $\mathcal{A}$, unless otherwise stated.

Definition. Let $U, W \subseteq FM(\mathcal{UF})$ and $u \in U$, let $\to$ be a binary relation on $FM(\mathcal{UF})$ and let $U \to W$. As for sets of formulas, U is said to be satisfiable if $\exists u(u \text{ is satisfiable})$. The relation $\to$ preserves satisfiability if $\forall U \forall W (U \text{ is satisfiable iff } W \text{ is satisfiable})$.

Theorem 2. *Let $\bar{x}^s = \mathcal{V}ar(s)$, $\bar{x}^l = \mathcal{V}ar(l)$, $\bar{x} = (\mathcal{V}ar(p) \cup \mathcal{V}ar(r)) \setminus \bar{x}^l$ and $\bar{x}^\theta = \mathcal{VR}ange(\theta)$. For $\bar{y}, \bar{z} \subseteq \mathcal{V}$, let $B(\bar{y}, \bar{z}, \tau)$ denote the formula $\forall \bar{y}(\vee_\pi \exists \bar{z}(p\sigma\tau \wedge \bar{y} = \bar{y}\tau))$. If $\forall \rho \forall \pi$(formula $p \Rightarrow l = r$ is valid) and $\forall \rho$(formula $B(\bar{x}^s \cup \bar{x}^l, \bar{x} \cup \bar{x}^\theta, \theta)$ is valid), then $\to_R$ preserves satisfiability.*

Proof. Let $\bar{x}^\sigma = \mathcal{V}\mathcal{R}ange(\sigma)$, $\bar{x}^A = \mathcal{V}ar(A)$ and $\bar{x}^\phi = \mathcal{V}\mathcal{R}ange(\phi)$. The conclusion immediately follows from the following two claims.

Claim 1. If $B(\bar{x}^s \cup \bar{x}^l, \bar{x} \cup \bar{x}^\theta, \theta)$ is valid, then $B(\bar{x}^A, \bar{x} \cup \bar{x}^\phi, \phi)$ is valid.

Claim 2. Let $\forall\pi(p \Rightarrow l = r$ is valid) and $B(\bar{x}^A, \bar{x} \cup \bar{x}^\phi, \phi)$ is valid. Then A is satisfiable iff $\{(p\sigma \wedge A[r\sigma]_q)\phi|\pi\}$ is satisfiable.

Proof of Claim 1. Let γ be an assignment. It is sufficient to prove that there is a branch π and an assignment γ''' such that $\gamma'''_{|\bar{x}^A} = \gamma_{|\bar{x}^A}$, $\gamma'''(\bar{x}^A) = \gamma'''(\bar{x}^A\phi)$ and $\gamma'''(p\sigma\phi) = \mathit{true}$. Let $Z = \bar{x}^A \cup \bar{x} \cup \bar{x}^\theta$. For γ, there is an assignment γ' such that γ' coincides with γ everywhere except $\bar{x}^s$ and $\gamma'(\bar{x}^s) = \gamma'(\bar{x}^s\sigma)$. By the premise of the claim, there is a branch π and an assignment γ'' such that γ'' coincides with γ' everywhere except $\bar{x} \cup \bar{x}^\theta$, $\gamma''(\bar{x}^s) = \gamma''(\bar{x}^s\theta)$, $\gamma''(\bar{x}^l) = \gamma''(\bar{x}^l\theta)$ and $\gamma''(p\sigma\theta) = \mathit{true}$. Thus

$$\gamma''(\bar{x}^s\sigma) = \gamma'(\bar{x}^s\sigma) = \gamma'(\bar{x}^s) = \gamma''(\bar{x}^s) = \gamma''(\bar{x}^s\theta)$$

and

$$\gamma''(\bar{x}^l\sigma) = \gamma'(\bar{x}^l\sigma) = \gamma'(\bar{x}^l) = \gamma''(\bar{x}^l) = \gamma''(\bar{x}^l\theta).$$

Since $\mathcal{D}om(\sigma) \cup \mathcal{D}om(\theta) \subseteq \bar{x}^s \cup \bar{x}^l$, by the definition of trivial unifier, there is an assignment γ''' such that γ coincides with γ'' everywhere except $\bar{x}^\phi$, $\gamma'''(\bar{x}^l) = \gamma'''(\bar{x}^l\phi)$ and $\gamma'''(\bar{x}^A) = \gamma'''(\bar{x}^A\phi)$. So $\gamma'''_{|\bar{x}^A} = \gamma_{|\bar{x}^A}$, $\gamma'''(\bar{x}^A) = \gamma'''(\bar{x}^A\phi)$ and $\gamma'''(p\sigma\phi) = \gamma'''(p\sigma) = \gamma''(p\sigma) = \gamma''(p\sigma\theta) = \mathit{true}$. □

Proof of Claim 2. Suppose A is satisfiable. Hence there is an assignment γ such that $\gamma(A) = \mathit{true}$. By the definition of unifier,

$$A_q\phi = s\sigma\phi = s\theta\phi = l\theta\phi = l\sigma\phi$$

and, hence, $A\phi = A[l\sigma]\phi$. By the second premise of the claim, there is a branch π and an assignment γ' such that $\gamma'_{|\bar{x}^A} = \gamma_{|\bar{x}^A}$, $\gamma'(\bar{x}^A) = \gamma'(\bar{x}^A\phi)$ and $\gamma'(p\sigma\phi) = \mathit{true}$. So

$$\gamma(A) = \gamma'(A) = \gamma'(A\phi) = \gamma'(A[l\sigma]_q\phi) = \gamma'(A[r\sigma]_q\phi) = \gamma'((p\sigma \wedge A[r\sigma]_q)\phi).$$

Hence $(p\sigma \wedge A[r\sigma]_q)\phi$ is satisfiable. Suppose there is a branch π such that $(p\sigma \wedge A[r\sigma]_q)\phi$ is satisfiable. Hence there is an assignment γ such that $\gamma(A[r\sigma]_q)\phi = \mathit{true}$ and $\gamma(p\sigma\phi) = \mathit{true}$. By the first premise of the claim, $\gamma(A[r\sigma]_q\phi) = \gamma(A[l\sigma]_q\phi) = \gamma(A\phi)$. For γ, there is an assignment γ' such that γ' coincides with γ everywhere except $\bar{x}^\sigma$ and $\gamma'(\bar{x}^\sigma) = \gamma'(\bar{x}^\sigma\phi)$. Hence $\gamma'(A) = \gamma'(A\phi) = \mathit{true}$ and A is satisfiable. □

5. Relation between termination and simplification orderings

A relation between termination of term rewriting systems and simplification orderings can be found in [8]. We would like to prove similar result for our formalism.

Definition. Let $*$ denote the logical connectives $\Rightarrow$, $\vee$ or $\wedge$ and $t, t' \in \mathcal{T}$. A map *Dec* is called an expression decomposition if (i) $Dec(true) = \emptyset$ and $Dec(false) = \emptyset$; (ii) $\forall t \forall t' (Dec(t = t') = \{t, t'\})$; (iii) $Dec(\neg A) = Dec(A)$ and $\forall * (Dec(A * B) = Dec(A) \cup Dec(B))$. Let $\succ$ be a strict partial ordering. R decreases w.r.t. $\succ$ if $\forall \rho \forall \pi \ (Dec(l) \succ_m Dec(r) \cup Dec(p))$. The branch π is uniform if $l = s$. The rule ρ is uniform if $\forall \pi (\pi$ is uniform). R is uniform if $\forall \rho$ (ρ is uniform).

Theorem 3. *Let R be a uniform FRS and $\succ$ be simplification ordering stable w.r.t. substitutions. R is terminating if $\exists \succ$ (R decreases w.r.t. $\succ$).*

Proof. Let $\succ_n = \{(u, v) | Dec(u) \succ_m Dec(v)\}$. In the proof we will use the following claim.

Claim. $\forall u \forall u' \exists \mu \ (u (\rightarrow_R)_b u' \Rightarrow u \succ_n u' \mu)$.

Suppose, for a proof by contradiction, that R is not terminating. Since R is not terminating, there is an infinite chain $u_0 (\rightarrow_R)_b u_1 (\rightarrow_R)_b \dots$. By the claim, there is an infinite sequence $\mu_1, \mu_2, \dots$ of renamings such that $u_{i-1} \succ_n u_i \mu_i$ for each $i \geq 1$. Let $u'_0 = u_0$ and $u'_i = u_i \mu_i \mu_{i-1} \dots \mu_1$ for each $i \geq 1$. Since $\succ$ is stable w.r.t. substitutions, $\succ_n$ is stable w.r.t. substitutions and, hence, $u'_{i-1} \succ_n u'_i$ for each $i \geq 1$. By the definition of simplification ordering, $\succ$ is well-founded and, hence, $\succ_n$ is well-founded. However, this contradicts the existence of infinite chain $u_0 \succ_n u_1 \succ_n \dots$. Hence R is terminating.

Proof of Claim. Let $A[s\sigma]_q (\rightarrow_R)_b (p\sigma \wedge A[r\sigma]_q)\phi$. It is sufficient to prove that $\exists \mu (A[s\sigma]_q \succ_n (p\sigma \wedge A[r\sigma]_q)\phi\mu)$. By the definition of ϕ and uniformness, there is a renaming τ such that $\phi = \sigma\tau$. Thus $A\phi = A\sigma\tau = A\tau$ and $(p\sigma \wedge A[r\sigma]_q)\phi = (p\sigma \wedge A[r\sigma]_q)\tau$. Since R decreases w.r.t. $\succ$, we have $Dec(l) \succ_m Dec(p) \cup Dec(r)$. By the definition of simplification ordering, $\succ$ is stable w.r.t. substitutions and has a subterm property.

So $Dec(A[s\sigma]_q) \succ_m Dec(p\sigma) \cup Dec(A[r\sigma]_p)$ and the conclusion of the claim is obtained for $\mu = \tau^{-1}$. □

6. Selector elimination systems

For $\mathcal{C} \subseteq \mathcal{F} \cup \mathcal{P}$, $\mathcal{E}(\mathcal{C})$ ($\mathcal{T}(\mathcal{C})$) denotes the set of all expressions (terms) built up only from symbols of the set $\mathcal{C} \cup \mathcal{V}$.

Many FRSs arising in practice are constructor FRSs. A constructor FRS R is a FRS in which the set of function and predicate symbols can be partitioned into a set $\mathcal{D}$ of selectors and a set $\mathcal{C}$ of constructors, such that for every rule $r \in R$ its sample has the form $f(t_1, \dots, t_n)$ with $f \in \mathcal{D}$ and $t_1, \dots, t_n \in \mathcal{T}(\mathcal{C})$. Note that the related concept for term rewriting systems is considered, for instance, in [6]. The following class of constructor FRSs (selector elimination systems or SESs for short) allows us to design simplification procedures eliminating selectors. The idea of such systems is percolating selectors through constructors down to variables followed by their elimination by replacement of variables. Unfortunately, even very strong restrictions imposed on AES guarantee the termination only w.r.t. a certain strategy.

Throughout the rest of the paper, R is a constructive FRS w.r.t. a set $\mathcal{C}$ of constructors and a set $\mathcal{D}$ of selectors. Let us first introduce some concepts which allow us to analyze a structure of expressions w.r.t. $\mathcal{C}$ and $\mathcal{D}$.

Definition. Let $X \subseteq \mathcal{V}$ and $x, y \in X$. An expression u is constructive if $u \in \mathcal{E}(\mathcal{C})$. A substitution σ is constructive if $\forall z(z\sigma \in \mathcal{T}(\mathcal{C}))$. An expression u is linear if $\forall z(|\mathcal{O}(z, \mathcal{MV}ar(u))| \leq 1)$. A substitution σ is linear on X if $\forall x, y(x\sigma$ is linear and $x \neq y \Rightarrow \mathcal{V}ar(x\sigma) \cap \mathcal{V}ar(y\sigma) = \emptyset)$. Let $q_1, q_2 \in \mathcal{P}(u)$ and $q_2 \neq q_1$, and let q_2 be a prefix of q_1. An expression u is embedded if $\exists q_1 \exists q_2$ $(root(u_{q_1}), root(u_{q_2}) \in \mathcal{D})$. An expression u is simple if u is not embedded. An expression u is a call if $root(u) \in \mathcal{D}$.

Let $C_m(u)$ denote the multiset of all simple calls of u.

Definition. A map μ_c is a constructor measure if $\mu_c(u) = \{|t| | t \in C_m(u)\}$. A map Dec_v is a variable decomposition if $Dec_v(u) = \cup_{t \in C_m(u)} \mathcal{MV}ar(t)$.

Let us now define selector elimination systems.

Definition. A branch π is percolating if π is uniform, p and r are simple, $Dec_v(l) \supseteq Dec_v(p) \cup Dec_v(r)$, $\mu_c(l) >_m \mu_c(r)$ and $\mu_c(l) >_m \mu_c(p)$. A branch π is eliminating if $\forall\theta(\theta$ is linear on $\mathcal{V}ar(s)$ and constructive), p and r are constructive. Let t be a simple call. An FRS R is a selector elimination system if $\forall t(t$ is a redex of $R)$ and $\forall\rho\forall\pi(\pi$ is either percolating or eliminating).

As mentioned above, SESs are not terminating in the general case.

Example. Let the FRS R consist of the following rules:

$r_1 : (\{f(c(z)) \rightarrow z\}, f(x))$,

$r_2 : (\{f(c(z)) \to z\}, f(c(x)))$,
$r_3 : (\{f(d(x,y)) \to d(f(y), f(x))\}, f(d(x', y')))$,
$r_4 : (\{h(c(z)) \to d(h(z), z)\}, h(c(x)))$,
$r_5 : (\{h(d(x,y)) \to x\}, h(d(x', y')))$,
$r_6 : (\{h(d(x,y)) \to x\}, h(z))$.

We prove first that R is a SES. Let t be a simple call. By the definition of R, t has one of the following forms: $f(\_)$, $f(c(\_))$, $f(d(\_,\_))$, $h(\_)$, $h(c(\_))$ or $h(d(\_,\_))$. It is obvious that t is a redex of R. Since $\theta = (x \to c(z))$ is constructive and linear on $\{x\}$ and $s = f(x)$ is constructive, $\{r_1\}$ is a SES. Since $\pi = f(d(x,y)) \to d(f(y), f(x))$ is uniform, $s = f(d(x,y))$ is simple, $\{x,y\} \supseteq \{x,y\}$ and $\{2\} >_m \{1,1\}$, $\{r_3\}$ is a SES. For other rules the proof is analogous. Therefore R is a SES.

However R is not terminating, since the chain $\underline{d(f(x), f(h(x)))} \to_{r_1} d(z, f(h(c(z)))) \to_{r_4} d(z, f(d(h(z), z))) \to_{r_3} d(z, \underline{d(f(z), f(h(z)))}) \to_{r_1} \ldots$ is infinite.

However, SESs are terminating w.r.t. the any-innermost strategy.

Theorem 4. *Let R be a selector elimination system. Then R is terminating w.r.t. the any-innermost strategy.*

Proof. Theorem 4 is an immediate corollary of Theorem 5 which we shall consider below. □

7. Selector elimination systems with argument status

Consider the generalization of SESs (SESs with argument status) which also has the termination property w.r.t. the any-innermost strategy.

Example. Let $R = (\{f(g(g(x)), y) \to f(x, f(a, y))\}, f(g(g(x)), y))$. It is obvious that R is terminating w.r.t. the any-innermost strategy. However R is not a SES, since the term $f(x, f(a, y))$ is embedded.

The problem of extending SESs to examples of such kind can be decided if we take into account only certain arguments of function symbols. In this example the term $f(x, f(a, y))$ is simple if we take into account only the first argument of f. Let us introduce the corresponding definition.

Definition. A map $\mathcal{A} : \mathcal{D} \to \mathcal{P}(N)$ is an argument status if $\mathcal{A}(f) \subseteq \{1, \ldots, \mathcal{A}r(f)\}$, where $\mathcal{A}r(f)$ is an arity of f.

The definitions of embedded expression, call, constructor measure and variable measure are modified to take into account an argument status.

Definition. Let $u_q = f(t_1, \dots, t_n)$ and $i \in \mathcal{A}(f)$. An expression u is embedded w.r.t. $\mathcal{A}$ if $\exists q \exists i(f \in \mathcal{D}$ and t_i is not constructive). An expression u is simple w.r.t. $\mathcal{A}$ if u is not embedded w.r.t. $\mathcal{A}$.

Let $C_m(u)$ denote the multiset of all simple (w.r.t. $\mathcal{A}$) calls of u and for $t = f(t_1, \dots, t_n) \in \mathcal{E}$, $I(t)$ denote the multiset $\{t_i | i \in \mathcal{A}(f)\}$.

Definition. Let $t \in C_m(u)$ and $t' \in I(t)$. A map μ_c is a constructor measure if $\forall u(\mu_c(u) = \{\sum_{t'} |t'| \mid t\})$. A map Dec_v is a variable decomposition if $\forall u(Dec_v(u) = \cup_t \cup_{t'} \mathcal{M}\mathcal{V}ar(t'))$.

Let us define the selector elimination systems with argument status.

Definition. A branch π is percolating w.r.t. $\mathcal{A}$ if π is uniform, p and r are simple w.r.t. $\mathcal{A}$, $Dec_v(l) \supseteq Dec_v(p) \cup Dec_v(r)$, $\mu_c(l) >_m \mu_c(p)$ and $\mu_c(l) >_m \mu_c(r)$. Let $s' \in I(s)$. A branch π is eliminating w.r.t. $\mathcal{A}$ if p and r are constructive and $\forall \theta(\theta$ is linear on $\mathcal{V}ar(s)$ and constructive and $\mathcal{V}ar(s) \cap \mathcal{D}om(\theta) \subseteq \cup_{s'} \mathcal{V}ar(s'))$. Let t be a simple call. An FRS R is a selector elimination systems with argument status $\mathcal{A}$ if $\forall t(t$ is a redex of $R)$ and $\forall \rho \forall \pi(\pi$ is either percolating w.r.t. $\mathcal{A}$ or eliminating w.r.t. $\mathcal{A})$.

Theorem 5. *Let R be a SES with an argument status $\mathcal{A}$. Then R is terminating w.r.t. the any-innermost strategy.*

Proof. Let $C_e(u)$ denote the multiset of all embedded (w.r.t. $\mathcal{A}$) calls of u. Let $n = \mathcal{O}(x, Dec_v(u))$ and $n \neq 0$. A map μ_a is a selector measure if $\mu_a(u) = |C_e(u)|$. A map μ_v is a variable measure if $\mu_v(u) = \{n|x\}$. Let $\succ$ be a strict partial ordering on $\mathcal{E}$ such that for any expressions u and u', $u \succ u'$ iff $(\mu_a(u), \mu_v(u), \mu_c(u))$ is lexicographically bigger than $(\mu_a(u'), \mu_v(u'), \mu_c(u'))$ with orderings $>$, $>_m$ and $>_m$ on the first, second and third elements of the triple, correspondingly. □

Let $A[s\sigma]_q(\rightarrow_{Ri})_b(p\sigma \wedge A[r\sigma]_q)\phi$. First we state some lemmas (their proof does not depend on modification of the definition of formula rewriting systems and can be found in [4]):

Lemma 1. $\mu_a(A[s\sigma]_q) \geq \mu_a((p\sigma \wedge A[r\sigma]_q)\phi)$;

Lemma 2. *Let* $\mu_a(A[s\sigma]_q) = \mu_a((p\sigma \wedge A[r\sigma]_q)\phi)$. *Then* $\mu_v(A[s\sigma]_q) \geq_m \mu_v((p\sigma \wedge A[r\sigma]_q)\phi)$;

Lemma 3. *Let* $\mu_a(A[s\sigma]_q) = \mu_a((p\sigma \wedge A[r\sigma]_q)\phi)$ *and* $\mu_v(A[s\sigma]_q) = \mu_v((p\sigma \wedge A[r\sigma]_q)\phi)$. *Then* $\mu_c(A[s\sigma]_q) >_m \mu_c((p\sigma \wedge A[r\sigma]_q)\phi)$.

By the definition of lexicographical ordering, $\succ$ and $\succ_m$ are well-founded. By the definition of lexicographical ordering and lemmas 1–3, $A[s\sigma]_q \succ (p\sigma \wedge A[r\sigma]_q)\phi$ and, hence, R is terminating w.r.t. the any-innermost strategy.

8. Examples

Let us illustrate introduced concepts using simplification procedures for the tuple theory as an example. We use the letters x, y, z and w to denote tuples, the letters a, b and c to denote elements of tuples and the letters i, j and k to denote integers. The tuple theory includes the operations of concatenation $*$, tuple constructor $[.]$, empty tuple $[\,]$, size $|.|$, indexing $.(.)$ and slice $.(. .. .)$ such that $x * y$ yields the concatenation of x and y, $[a]$ yields the tuple consisting of only one element a, $[\,]$ yields the empty tuple, $|x|$ yields the length of x, $x(i)$ yields the i-th component of the tuple x and $x(i..j)$ yields the section or slice of x extending from its i-th through its j-th component, inclusive. In addition, we use the usual arithmetical operations $<$, $\leq$, $+$, $-$ and integer constant 1. We design the terminating elimination procedures for the indexing and slice operations.

8.1. Elimination procedure for indexing operation

We first design the simplification procedure which eliminates the terms of the form $(\_*\_)(\_)$. To percolate the indexing operation through the concatenation we must implement the case analysis: (i) If $i \leq |x|$, then $(x * y)(i) = x(i)$ and (ii) If $|x| < i$, then $(x * y)(i) = y(i)$. Let us note that the left-hand sides of the equalities appearing in (i) and (ii) coincide. Then the case analysis is implemented by the uniform formula rewriting rule with the base consisting of two conditional rules corresponding to the cases (i) and (ii). For example, the rule $i \leq |x| | (x * y)(i) \to x(i)$ corresponds to the case (i).

Let us check the sufficient conditions of preserving satisfiability for the rule. The first condition is reduced to checking the validity of the formulas $i \leq |x| \Rightarrow (x * y)(i) = x(i)$ and $|x| < i \Rightarrow (x * y)(i) = y(i)$ in the tuple theory. Let B denote the formula $x = x' \wedge y = y' \wedge i = i'$ where x', y' and i' are new variables. The most general unifiers of the left-hand sides of both branches of the rule and its sample have the same form $(x \to x', y \to y', i \to i')$. Thus the second condition is reduced to checking the validity of the following formula $\exists x' \exists y' \exists i' (i' \leq |x'| \wedge B) \ \vee \ \exists a' \exists i' (|x'| < i' \wedge B)$ in the tuple theory. After obvious simplifications the formula is reduced to the completeness condition $i \leq |x| \ \vee \ |x| < i$ of the case analysis. It is obvious that all these formulas are valid in the tuple theory.

We now prove that the branches of the rule are percolating. Let p denote $i \leq |x|$, l denotes $(x * y)(i)$ and r denotes $x(i)$. The first branch $p|l \rightarrow r$ of the rule is percolating if p and r are simple, $\mu_c(p) <_m \mu_c(l)$, $\mu_c(r) <_m \mu_c(l)$ and $Dec_v(p) \cup Dec_v(r) \subseteq Dec_v(l)$. After obvious simplifications we obtain the easily verified properties $\emptyset <_m \{1\}$, $\{0\} <_m \{1\}$ and $\emptyset \cup \{x, i\} \subseteq \{x, y, i\}$. The second branch of the rule is verified in the similar way.

The simplification procedures which eliminate the terms of the form $[.](\_)$ and $[\,](\_)$ are similarly designed. The first procedure applies the case analysis: (i) If $i = 1$, then $[a](i) = a$ and (ii) If $i \neq 1$, then $[a](i) = \omega$ (indeterminate value). The second procedure is based on the usual term rewriting rule $[\,](i) \rightarrow \omega$ simulated by the uniform formula rewriting rule with only one branch $(\{[\,](i) \rightarrow \omega\}, [\,](i))$.

Let us design the elimination procedure for the indexing operation. We can percolate the indexing operation through the concatenation, tuple constructor and empty tuple by the above uniform rules. Therefore we need only the rule which eliminates the terms of the form $x(\_)$, where x is a variable. This rule is also based on the case analysis: (i) If $x = y * [a] * z$ and $|y|+1 = i$, then $x(i) = a$ and (ii) If $i < 1 \vee |x| < i$, then $x(i) = \omega$. The base of the rule consists of two conditional rules corresponding to the cases (i) and (ii). However, this correspondence takes into account the possible structure of x. For example, the rule $|y| + 1 = i|(y * [a] * z)(i) \rightarrow a$ corresponds to the case (i). The sample of the rule is the term $x(i)$.

We first check the sufficient conditions of preserving satisfiability for the rule. The first condition is reduced to checking the validity of the formulas $|y| + 1 = i \Rightarrow (y * [a] * z)(i) = a$ and $(i < 1 \vee |x| < i) \Rightarrow x(i) = \omega$ in the tuple theory. The most general unifiers of the left-hand sides of first and second branches of the rule and its sample have the form

$$(x \rightarrow y' * [a'] * z', y \rightarrow y', z \rightarrow z', a \rightarrow a', i \rightarrow i')$$

and $(x \rightarrow x', i \rightarrow i')$, respectively. Thus the second condition is reduced to checking the validity of the formula

$$\begin{gathered}\exists a' \exists y' \exists z' \exists i' (|y'| + 1 = i' \wedge a = y' * [a'] * z' \wedge y = y' \wedge z = z' \wedge i = i') \\ \vee \\ \exists x' \exists i' ((i' < 1 \vee |x'| < i') \wedge x = x' \wedge i = i')\end{gathered}$$

in the tuple theory. After obvious simplifications the formula is reduced to the completeness condition $\exists a \exists y \exists z (|y| + 1 = i \wedge x = y * [a] * z) \vee i < 1 \vee |x| < i$ of the case analysis. It is obvious that all these formulas are valid in the tuple theory.

We now prove that the branches of the rule are eliminating. The first branch $|y| + 1 = i|(y * [a] * z)(i) \rightarrow a$ of the rule is eliminating if $|y| + 1 = i$ and a are constructive and the most general unifier

$$\theta = (x \to y' * [a'] * z', y \to y', z \to z', a \to a', i \to i')$$

of $(y * [a] * z)(i)$ and $x(i)$ is linear on $\{x, i\}$ and constructive. The check of these properties is straightforward. The second branch of the rule is verified in the similar way.

Let us demonstrate that the formula rewriting system $\mathcal{R}$ consisting of all the above rules is terminating w.r.t. the any-unnermost strategy. By Theorem 4, it is sufficient to prove that $\mathcal{R}$ is a selector elimination system. It is obvious that R is a constructive system, where the indexing operation is a selector and the rest operations are constructors. For some branches of the rule of $\mathcal{R}$ we have proved that they are either percolating or eliminating. For other branches the proof is performed similarly. A simple call w.r.t. the constructive system $\mathcal{R}$ has one of the following four forms $(u * u')(v)$, $[u](v)$, $[\,](v)$ or $x(v)$, where u and v are constructive terms. Since for each of these forms there is a corresponding rule in $\mathcal{R}$, any simple call is a redex of $\mathcal{R}$. Thus R is a selector elimination system.

8.2. Elimination procedure for slice operation

The elimination procedure for the slice operation is designed similarly. The formula rewriting system $\mathcal{R}'$ underlying the procedure consists of three rules percolating the operation through concatenation, tuple constructor and empty tuple and the fourth rule eliminating it. The sufficient conditions of preserving satisfiability are verified for $\mathcal{R}'$ in the same manner as for $\mathcal{R}$. Let us prove the system $\mathcal{R}$ is terminating w.r.t. the any-innermost strategy. It is obvious that R is a constructive system, where the slice operation is a selector and the rest of operations are constructors.

The rule percolating the slice operation through the concatenation implements the following case analysis: (i) If $j \leq |x|$, then $(x*y)(i..j) = x(i..j)$; (ii) If $|x| < i$, then $(x*y)(i..j) = y(i..j)$ and (iii) If $i \leq |x| < j$, then $(x*y)(i..j) = x(i..|x|) * y(1..j - |x|)$. Let p denote $i \leq |x| < j$, l denote $(x * y)(i..j)$ and r denote $x(i..|x|) * y(1..j - |x|)$. Consider the third branch $p|l \to r$ of the rule. The branch is not eliminating, since r is not constructive. The branch is percolating if p and r are simple, $\mu_c(p) <_m \mu_c(l)$, $\mu_c(r) <_m \mu_c(l)$ and $Dec_v(p) \cup Dec_v(r) \subseteq Dec_v(l)$. After obvious simplifications we obtain the properties $\emptyset <_m \{1\}$, $\{1, 3\} <_m \{1\}$ and $\emptyset \cup \{x, x, x, y, i, j\} \subseteq \{x, y, i, j\}$. It is obvious that the first and third of these properties are false. Thus $\mathcal{R}'$ is not a selector elimination system. This is due to the fact that we take into account variables and constructors appearing in the second and third arguments of the slice operation in spite of the fact that we percolate the slice operation only with respect to the first argument. To avoid this, we will use the argument status $\mathcal{A} = \{1\}$. Then, after simplifications, the properties $\mu_c(r) <_m \mu_c(l)$ and $Dec_v(p) \cup Dec_v(r) \subseteq Dec_v(l)$ take the easily proved form $\{0, 0\} <_m \{1\}$ and $\emptyset \cup \{x, y\} \subseteq \{x, y\}$. Thus the branch is eliminating w.r.t. $\mathcal{A}$. The fact that the other uniform branches of the rules of $\mathcal{R}'$ are also percolating w.r.t. $\mathcal{A}'$ is verified similarly.

It is not difficult to check that any simple call is a redex of $\mathcal{R}'$. It remains to prove that the branches of the fourth rule are eliminating. This rule is also based on the case analysis: (i) If $x = y * w * z$, $|y| + 1 = i$, $|y| + |w| = j$ and $1 \leq |w|$, then $x(i..j) = w$ and (ii) If $i < 1 \vee |x| < j \vee j < i$, then $x(i..j) = \omega$. The base of the rule consists of two conditional rules corresponding to the cases (i) and (ii). This correspondence also takes into account the possible structure of variable x. For example, the conditional rule $|y| + 1 = i \wedge |y| + |w| = j \wedge 1 \leq |w| | (y * w * z)(i) \rightarrow w$ corresponds to the case (i). The sample of the rule is the term $x(i..j)$. The first branch of the formula rewriting rule is eliminating if $|y| + 1 = i \wedge |y| + |w| = j \wedge 1 \leq |w|$ and w are constructive w.r.t. $\mathcal{A}$ and the most general unifier

$$\theta = (x \rightarrow y' * w' * z', y \rightarrow y', z \rightarrow z', a \rightarrow a', i \rightarrow i')$$

of $(y * w * z)(i..j)$ and $x(i..j)$, where y', z', w', i' and j' are new variables, is linear on $\{x, i, j\}$ and constructive. The check of these properties is straightforward. The second branch of the rule is verified in the similar way. Thus $\mathcal{R}'$ is a selector elimination system with the argument status $\mathcal{A}$ and, hence, by Theorem 5, $\mathcal{R}'$ is terminating w.r.t. the any-innermost strategy.

Conclusion

The paper presents the method of designing simplification procedures based on formula rewriting systems which includes the following new features:

(i) the concept of formula transformation correctness which preserves formula satisfiability rather than formula equivalence;

(ii) the generalized concept of formula rewriting rules exploiting generalized conditional term rewriting (in particular, with extra variables);

(iii) the tools for proving termination of formula rewriting systems.

Proving verification conditions which appear in program verification is an important area of application of the method. Proving correctness conditions is performed in interactive mode in most of the verification systems. Our method allows the proof of correctness conditions to be done automatically. At present in the framework of program verification system SPECTRUM ([9, 10]) we are implementing a new prover based in part on the method. Some details of the theory of formula rewriting systems and their application to problem-oriented verification have been considered in [3, 4]. In particular, proving correctness conditions of simple array sorting programs and elimination of such data structures as array, sequential files and lists are described. An experiment on automatic verification of the program of file sorting by natural merge has been performed in the framework of the project SPECTRUM by means of this method.

Acknowledgements. I would like to thank my scientific supervisor Valery Nepomniaschy for very helpful discussions. I am also indebted to Nikolai Shilov for detailed comments and many useful suggestions which considerably improved style and contents of this paper.

References

[1] H. Kirchner, *On the use of constraints in automated deduction*, Lecture Notes in Computer Science, **910**, 1995, 128–146.

[2] H. Comon, *Constraints in term algebras. An overview of constraint solving techniques*, Lecture Notes in Computer Science, **355**, 1995, 109–120.

[3] I. S. Anureev, *Integrated term rewriting rules and their application to automatic program verification*, Problems of specification and verification of concurrent systems, Institute of Informatics Systems, Novosibirsk, Russia, 1995, 185–213 (in Russian).

[4] I. S. Anureev, *Formula rewriting systems*, Report 40, Institute of Informatics Systems, Novosibirsk, Russia, 1997 (in Russian).

[5] N. Dershowitz and J.-P. Jouannaud, *Rewrite systems*, Handbook of Theoretical Computer Science, B(6), 1990, 243–320.

[6] J. W. Klop, *Term rewriting systems*, Handbook of Logic in Computer Science, **2**, 1993. 1–116.

[7] M. Dauchet, *Simulation of Turing machines by a left-linear rewrite rule*, Lecture Notes in Computer Science, **910**, 62–67.

[8] J. Steinbach, *Simplification ordering: history of results*, Fundamenta Informaticae, **24(1)**, 1995, 47–87.

[9] V. A. Nepomniaschy and A. A. Sulimov, *Problem-oriented means of program specification and verification in project SPECTRUM*, Lecture Notes in Computer Science, **722**, 1993, 374–378.

[10] V. A. Nepomniaschy and A. A. Sulimov, *Problem-oriented verification system and its application to linear algebra programs*, Theoretical Computer Science, **119**, 1993, 173–185.