Formula rewriting systems and their application to automated program verification*

I.S. Amrvev

We describe a notion of formula rewriting systems (FRSs) and investigate a relation between FRSs and techniques of term rewriting systems (TRSs) and narrowing. We demonstrate the use of FRSs for finding errors in array and file operations.

Introduction

The achievement of software reliability is an important problem of programming technology. The classical program verification based on the papers of Floyd and Hoare plays an essential role in deciding the problem. In this approach the program correctness (the correspondence between a program and its specification) is reduced to the validity of a set of formulas (verification conditions) of a specification language. In real programs the verification conditions can be formulas of a considerable length and their proving is arduous and needs automatization. Therefore the development of proving methods oriented to automated program verification is an urgent problem of the verification theory.

In the paper, the formalism of FRSs [2, 4] recently suggested is considered as one of such methods. FRSs combine the elements of TRSs with narrowing and represent a powerful means for the development of simplification procedures preserving satisfiability (a satisfiable (unsatisfiable) formula rewrites to a satisfiable (unsatisfiable) one). The use of FRSs is demonstrated by the examples of incorrect application of array and file operations.

1. Formula rewriting systems as a generalization of term rewriting systems and narrowing

As mentioned above, FRSs are a formula rewriting technique integrating the elements of TRSs and narrowing and preserving satisfiability. To understand the technique better, we introduce the notions of a formula rewriting rule and a reduction relation (generated by it) by choosing and generalizing some elements of TRS and narrowing by stages.

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 $\Sigma$-terms, $\mathcal{UF}$ denotes the set of unquantified $\Sigma$-formulas with equality $=$, $\mathcal{E} = \mathcal{T} \cup \mathcal{UF}$ denotes the set of $\Sigma$-expressions and $\mathcal{S}$ denotes the set of substitutions over $\mathcal{E}$.

For $u \in \mathcal{E}$ and $\sigma \in \mathcal{S}$, $\text{Var}(u)$ denotes the set of variables of $u$, $\mathcal{P}(u)$ denotes the set of positions of $u$ with $\Lambda$ as the topmost position, $\text{Dom}(u)$ denotes the domain of $u$ and $\text{VarRange}(\sigma)$ denotes the variable range of $\sigma$.

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

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

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

Let $\text{Var}(w)$ denotes a set of variables of a term (formula and so on) $w$. A term rewriting rule $\rho$ is a pair $l \rightarrow r$ of terms such that $\text{Var}(r) \subseteq \text{Var}(l)$ and $l \not\in \mathcal{V}$. A reduction relation $\rightarrow_{\rho}$ and a narrowing

*Partially supported by INTAS-RFBR under Grant 95-0378.
relation \( \sim_\rho \) generated by \( \rho \) are defined as follows: \( A[l]_q \to_\rho A[r]_q \) for all \( A \) and \( q \) such that \( l \sigma = t \) (\( t \) is an example of \( l \)) and \( A[l]_q \to_\rho A[r]_q \theta \) for all \( A \) and \( q \) such that \( l \) and \( t \) are unifiable and \( \theta \) is a most general unifier (MGU) of \( l \) and \( t \).

Step 1 (The restriction on \( \iota \)). It is difficult to define static conditions preserving satisfiability for narrowing relation primarily because the term \( \iota \) is chosen in an arbitrary way. As the rule \( \rho \) we take a pair \( (l \to r, s) \), where \( s \in T \) and \( s \) and \( \iota \) are unifiable. We shall consider only the terms \( \iota \) which are examples of \( s \). A reduction relation \( \to_\rho \) is defined as follows: \( A[l]_q \to_\rho A[r]_q \phi \) for all \( A \) and \( q \) such that \( \iota \) is an example of \( s \); \( \iota \) and \( t \) are unifiable and \( \phi \) is an MGU of \( l \) and \( t \).

Step 1'. (Another form of \( \to_\rho \)). Using the fact that \( \sigma \) and \( l \) are unifiable, the reduction relation \( \to_\rho \) can be rewritten into \( A[l]_q \to_\rho A[r]_q \phi \), where \( t = s \sigma, \phi \) is an MGU of substitutions \( \sigma \) and \( \sigma' \) can be defined in two equivalent ways:

- algorithmic: \( \phi \) is an MGU found by the unification algorithm without applying the term reduction rule;
- semantic: Let \( \bar{z} = \text{Dom}(\sigma) \cup \text{Dom}(\sigma'), \bar{z} = \text{Var}(z \sigma = z \sigma') \) and \( \psi = \forall \text{Range}(\phi) \). \( \phi \) is an MGU such that the formula \( \forall \bar{z} \exists \bar{y}(z \sigma = z \sigma' \Rightarrow \bar{z} = \bar{z} \phi) \) is valid.

Let us restrict the relation \( \to_\rho \) by the substitutions \( \phi \) which are trivial unifiers \( \sigma \) and \( \theta \).

Step 3. (The transition to conditional rules). A natural generalization of \( \rho \) is the transition from usual term rewriting rules to conditional ones. Let \( \rho = (p|l \to r, s) \), where \( p \) is an arbitrary quantified formula. The reduction relation \( \to_\rho \) is defined as follows: \( A[l]_q \to_\rho (p \sigma \wedge A[r]_q \phi) \) at the same conditions on \( A, q \) and \( \phi \).

Step 4. (The case analysis introduction). Another generalization \( \rho \) is the transition from one conditional term rewriting rule to a finite set of conditional rules (conditional term rewriting systems or CTRS for short) \( B \). Let \( p_\pi, l_\pi, r_\pi \) denote the premise, left-hand side and right-hand side of the rule \( \pi \in B \), correspondingly. The rule \( \rho \) is defined by the pair \( (B, s) \) such that \( s \) and \( l_\pi \) are unifiable for each \( \pi \in B \). Let \( \theta_\pi \) be an MGU of \( s \) and \( l_\pi \) for each \( \pi \in B \). The reduction relation \( \to_\rho \) is defined as follows:

\[
A[l]_q \to_\rho U = \{ (p_\pi \sigma \wedge A[l_\pi]_q \phi_\pi | \pi \in B) \}
\]

for all \( A \) and \( q \) such that \( l_\pi = \bar{s}, \phi_\pi \) is a trivial unifier of \( \sigma \) and \( \theta_\pi \) for each \( \pi \in B \). The formula multiset \( U \) is satisfiable iff one of the formulas of \( U \) is satisfiable. Thus the generalization allows us to perform a case analysis.

Step 5. (The abandonment of restrictions on CTRSs). The last step consists in the abandonment of restrictions usually imposed on CTRSs. We consider \( B \) as a finite set of triples \( p|l \to r \).

Now we can define a formula rewriting system. A formula rewriting rule \( \rho \) is a pair \( (B, s) \) such that for all \( \pi \in B \) terms \( l_\pi \) and \( s \) are unifiable. The set \( B \) is called the base of \( \rho \), \( \pi \in B \) are called the branches of \( \rho \), and the term \( s \) is called the skeleton of \( \rho \). The reduction relation generated by \( \rho \) is defined as at step 4 above. A formula rewriting system is defined as a finite set of formula rewriting rules. The reduction relation generated by an FRS is a union of reduction relations generated by all rules of the FRS.

2. The properties of formula rewriting systems

All notions (termination, normal form, redex and so on) of the theory of TRSs are easily propagated onto FRSs. We specially consider only the sufficient conditions of preserving satisfiability for the reduction relations generated by FRSs. The theorem about preserving satisfiability for a rule \( \rho \) have the following form.
Theorem 2.1. Let $A$ be an algebraic $\Sigma$-structure,
\[ z \in \text{Var}(x) \setminus \bigcup_{\pi \in \mathcal{B}} \text{Var}(\pi), \]
\[ \text{HN} \rightarrow (\bigcup_{\pi \in \mathcal{B}} (\text{Var}(\pi_x) \cup \text{Var}(\pi_y)) \setminus \text{Var}(\pi_z)) \cup \text{Var}(\pi), \]
\[ z = \forall \text{Range}(\theta). \]

If the formulas
(i) $\mu \rightarrow \pi \rightarrow \tau \tau$ for each $\pi \in \mathcal{B};$
(ii) $\forall \pi \forall \pi \forall \pi \forall x \forall y \forall z (\mu = \theta \land z = y \theta)$
are valid in $A$, then $\tau_\theta$ preserves satisfiability in $A$.

The proof of the theorem can be found in [4].

Consider an example of using FRs for elimination of functional symbols. It illustrates the notions of a FRs and a reduction relation (generated by it) and the sufficient conditions of the above theorem.

Example 2.2. Let the FR $R$ consist of rules $(f(g(x)) \rightarrow x), f(y))$ and $(f(g(x)) \rightarrow x), f(g(x)))$. MGUs $\theta_1$ and $\theta_2$ for rules 1 and 2 have the form $(x \rightarrow z, y \rightarrow g(z))$ and $(x \rightarrow z)$, correspondingly. The theorem conditions for rule 1 take the form $f(g(x)) = x$ and $\forall x \forall y \forall z (z = x \land y = g(z))$. For rule 2 the theorem conditions take the form $f(g(x)) = x$ and $\forall x \forall z (z = x)$. After obvious simplification the second conditions for the rules are reduced to $\forall x \forall y (x = g(y))$ and true, correspondingly.

Elimination of $f$ from the formula $f(x) = f(y)$ is performed in the following way: $f(x) \rightarrow R$ $z = f(g(x)) \rightarrow R$ $z = z$.

3. Finding errors in array and file operations

Throughout the rest of the paper, we use the letters $x$, $y$ and $z$ to represent tuples, the letters $u$, $v$ and $w$ to represent elements of tuples and the letters $i$, $j$, $l$ and $r$ to represent integers. Let $\omega$ denote an indeterminate value. Let $x \cup y$, $(u)$, and $\langle \rangle$ denote a concatenation of tuples $x$ and $y$, a singleton tuple and an empty tuple, correspondingly.

To demonstrate the possibilities of FRs for deciding the problems of automated program verification, we consider the examples of verification of the programs incorrectly applying array and file operations.

Example 3.1. An array $a$ is defined by a triple $(x, l, r)$, where array is an array constructor, tuple $x$ consists of the elements of the array $a$, integers $l$ and $r$ specify the left and right bounds of the array $a$, respectively. As usually, $a[i]$ denotes selection of the $i$-th element of $a$.

Let us check correctness of the following annotated program
\[ \{ l \leq i \leq r \} ; a[i] \in (\omega). \]

The verification condition of the program has the form $l \leq i \leq r \Rightarrow a[i] = \omega$. The verification condition is valid iff its negation $l \leq i \leq r \land a[i] = \omega$ is not satisfiable. To simplify its negation, we apply the rule
\[ [\{ A_1 \} \text{array}(x \cup (\omega) \cup y, l, r) [i] \rightarrow u, \]
\[ A_2 \text{array}(x \cup (\omega) \cup y, l, r) [i] \rightarrow \omega, \]
\[ A_3 \text{array}(x, l, r) [i] \rightarrow \omega, \]
\[ a[i]). \]

where
\begin{align*}
A_1 : & \quad u \neq \omega \land \text{len}(x) + \text{len}(y) - r - l \land l + 1 \land \text{len}(x), \\
A_2 : & \quad \text{len}(x) + \text{len}(y) = r - l \land l = l + \text{len}(x), \\
A_3 : & \quad l \leq r \land \text{len}(x) = r \land l + 1 \land \neg(l \leq i \leq r). 
\end{align*}

After applying the rule, we obtain the formula multiset
\{l \leq i \leq r \land A_l \land u = \omega, l \leq i \leq r \land A_r \land \omega = \omega, l \leq i \leq r \land A_\omega \land \omega = \omega\}.

It is satisfiable, since the formula \(l \leq i \leq r \land A_2 \land \omega = \omega\) is satisfiable. Therefore the verification condition is not valid if \(a\) satisfies the condition
\[ a = a \times y(x, y, u) \cup y, l, r \times A_1.\]

Example 3.2. A file \(f\) is defined by a tuple \(\text{file}(x, y, u)\), where \(\text{file}\) is a file constructor, tuples \(x\) and \(y\) consist of the read and unread elements of the file \(f\), \(u\) specifies the value of the buffer variable associated with the file \(f\).

Let us check the correctness of the following annotated program
\[ \{\text{true}\} f := \text{get}(f)\{f \neq \omega\}.\]

The verification condition of the program has the form \(\text{true} \Rightarrow \text{get}(f) \neq \omega\). To simplify its negation \(\text{true} \land \text{get}(f) = \omega\), we apply the rule
\[
\{\text{true} \land \text{file}(x, u) \cup y, u \Rightarrow \text{file}(x, y, u)\},
\text{file}(x, u, u) \Rightarrow \omega\},
\text{get}(f)\}.
\]

It defines the operational semantics of the Pascal-like operation \text{get}.

After applying the rule, we obtain the formula multiset
\[
\{\text{true} \land \text{file}(x, u) \cup y, u = \omega, \text{true} \land \omega = \omega\}.
\]

It is satisfiable, since the formula \(\text{true} \land \omega = \omega\) is satisfiable. Therefore the verification condition is not valid if \(f\) satisfies the condition
\[ f = \text{file}(x, \{\}, u).\]

The FR\(S\)s for the complete set of array and file operations are given in [3].

4. Conclusion

The paper presents the introduction to application of FR\(S\)s to automated program verification and includes the following features:

(i) the main definitions of the theory of FR\(S\)s;
(ii) the relation between FR\(S\)s and such rewrite techniques as TR\(S\)s and narrowing;
(iii) examples of using FR\(S\)s to check correctness of applying the array and file operations.

At present, in the framework of program verification system SPECTRUM [6, 7], we implement a new prover partially based on the FR\(S\)s. Some details of the theory of FR\(S\)s and their application to problem-oriented verification have been considered in [1, 9, 5]. In particular, the automated verification of several programs of text editing, array sorting and file sorting has been described.

References


