Joint NCC & IIS Bull., Comp. Science, 12 (1999), 1–5

Formula rewriting systems and their application to automated program verification*

I.S. Anureev

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 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, $\mathcal{P}(u)$ denotes the set of positions of u with Λ as the topmost 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 $1 \le i \le n$. In particular, $(\,)$ is an identity substitution.

Let $\mathcal{V}ar(w)$ denotes a set of variables of a term (formula and so on) w. A term rewriting rule ρ is a pair $l \to r$ of terms such that $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$ and $l \notin \mathcal{V}$. A reduction relation $\to_\rho$ and a narrowing

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

relation $\rightsquigarrow_\rho$ generated by ρ are defined as follows: $A[t]_q \to_\rho A[r\sigma]_q$ for all A and q such that $l\sigma = t$ (t is an example of l) and $A[t]_q \rightsquigarrow_\rho A[r]_q\theta$ for all A and q such that l and t are unifiable and θ is a most general unifier (MGU) of l and t.

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

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

Step 2. (The restriction on ϕ). Unfortunately, the restriction on t is not sufficient to define static conditions of preserving satisfiability for $\to_\rho$ because of ϕ chosen in an arbitrary way. The restriction on the class of MGUs is needed. For this purpose a special class of MGUs — trivial unifiers — is selected. A notion of a trivial unifier ϕ of substitutions σ and σ' can be defined in two equivalent ways:

– algorithmic: ϕ is an MGU found by the unification algorithm without applying the term reduction rule;

– semantic: Let $\bar{z} = \mathcal{D}om(\sigma) \cup \mathcal{D}om(\sigma')$, $\bar{x} = \mathcal{V}ar(\bar{z}\sigma = \bar{z}\sigma')$ and $\bar{y} = \mathcal{VR}ange(\phi)$. ϕ is an MGU such that the formula $\forall\bar{x}\exists\bar{y}(\bar{z}\sigma = \bar{z}\sigma' \Rightarrow \bar{x} = \bar{x}\phi)$ is valid.

Let us restrict the relation $\to_\rho$ by the substitutions ϕ which are trivial unifiers σ and θ.

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

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

$$A[t]_q \to_\rho U = \{(p_\pi\sigma \wedge A[r_\pi\sigma]_q)\phi_\pi | \pi \in \mathcal{B}\}$$

for all A and q such that $t\sigma = s$, ϕ_π is a trivial unifier of σ and θ_π for each $\pi \in \mathcal{B}$. 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 $\mathcal{B}$ as a finite set of triples $p|l \to r$.

Now we can define a formula rewriting system. A formula rewriting rule ρ is a pair $(\mathcal{B}, s)$ such that for all $\pi \in \mathcal{B}$ terms l_π and s are unifiable. The set $\mathcal{B}$ is called the base of ρ, $\pi \in \mathcal{B}$ are called the branches of ρ, and the term s is called the sample of ρ. The reduction relation generated by ρ 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 ρ have the following form.

Theorem 2.1. Let $\mathcal{A}$ be an algebraic Σ-structure,

$$\begin{aligned}
&\bar{x} = \mathcal{V}ar(s) \cup \bigcup_{\pi\in\mathcal{B}} \mathcal{V}ar(l_\pi),\\
&\bar{y}_\pi = (\bigcup_{\pi\in\mathcal{B}}((\mathcal{V}ar(r_\pi) \cup \mathcal{V}ar(r_\pi)) \setminus \mathcal{V}ar(l_\pi))) \setminus \mathcal{V}ar(s),\\
&\bar{z} = \mathcal{VR}ange(\theta).
\end{aligned}$$

If the formulas

(i) $p_\pi \Rightarrow l_\pi = r_\pi$ for each $\pi \in \mathcal{B}$;

(ii) $\forall \bar{x} \bigvee_{\pi\in\mathcal{B}} \exists \bar{y}_\pi \exists \bar{z}(p_\pi\theta_\pi \wedge \bar{x} = \bar{x}\theta_\pi)$

are valid in $\mathcal{A}$, then $\rightarrow_\rho$ preserves satisfiability in $\mathcal{A}$.

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

Consider an example of using FRSs 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 FRS R consist of rules $(\{f(g(x)) \rightarrow x\}, f(y))$ and $(\{f(g(x)) \rightarrow x\}, f(g(x)))$. MGUs θ_1 and θ_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 $\forall x \forall y \exists z(x = z \wedge y = g(z))$. For rule 2 the theorem conditions take the form $f(g(x)) = x$ and $\forall x \exists z(x = z)$. After obvious simplification the second conditions for the rules are reduced to $\forall x \exists y(x = g(y))$ and *true*, correspondingly.

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

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 ω denote an indeterminate value. Let $x \cup y$, $\langle u \rangle$, and $\langle\ \rangle$ denote a concatenation of tuples x and y, a singleton tuple and an empty tuple, correspondingly.

To demonstrate the possibilities of FRSs 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 $array(x, l, r)$, 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

$$\{l \le i \le r\}x := a[i]\{x \ne \omega\}.$$

The verification condition of the program has the form $l \le i \le r \Rightarrow a[i] \ne \omega$. The verification condition is valid iff its negation $l \le i \le r \wedge a[i] = \omega$ is not satisfiable. To simplify its negation, we apply the rule

$$\begin{aligned}
(\{&A_1|array(x \cup \langle u \rangle \cup y, l, r)[i] \rightarrow u,\\
&A_2|array(x \cup \langle \omega \rangle \cup y, l, r)[i] \rightarrow \omega,\\
&A_3|array(x, l, r)[i] \rightarrow \omega\},\\
&a[i]),
\end{aligned}$$

where

$$\begin{aligned}
A_1 :&\ \ u \ne \omega \wedge len(x) + len(y) = r - l \wedge i = l + len(x),\\
A_2 :&\ \ len(x) + len(y) = r - l \wedge i = l + len(x),\\
A_3 :&\ \ l \le r \wedge len(x) = r - l + 1 \wedge \neg(l \le i \le r).
\end{aligned}$$

After applying the rule, we obtain the formula multiset

$$\{l \le i \le r \wedge A_1 \wedge u = \omega, l \le i \le r \wedge A_2 \wedge \omega = \omega, l \le i \le r \wedge A_3 \wedge \omega = \omega\}.$$

It is satisfiable, since the formula $l \le i \le r \wedge A_2 \wedge \omega = \omega$ is satisfiable. Therefore the verification condition is not valid if a satisfies the condition

$$a = array(x \cup \langle\omega\rangle \cup y, l, r) \wedge A_2.$$

Example 3.2. A file f is defined by a triple $file(x, y, u)$, where $file$ 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

$$\{true\} f := get(f)\{f \ne \omega\}.$$

The verification condition of the program has the form $true \Rightarrow get(f) \ne \omega$. To simplify its negation $true \wedge get(f) = \omega$, we apply the rule

$$\begin{aligned}(\{&file(x, \langle u\rangle \cup y, v) \to file(x \cup \langle u\rangle, y, \omega),\\ &file(x, \langle\ \rangle, u) \to \omega\},\\ &get(f)).\end{aligned}$$

It defines the operational semantics of the Pascal-like operation get.

After applying the rule, we obtain the formula multiset

$$\{true \wedge file(x \cup \langle u\rangle, y, \omega) = \omega, true \wedge \omega = \omega\}.$$

It is satisfiable, since the formula $true \wedge \omega = \omega$ is satisfiable. Therefore the verification condition is not valid if f satisfies the condition

$$f = file(x, \langle\ \rangle, u).$$

The FRSs for the complete set of array and file operations are given in [3].

4. Conclusion

The paper presents the introduction to application of FRSs to automated program verification and includes the following features:

(i) the main definitions of the theory of FRSs;

(ii) the relation between FRSs and such rewrite techniques as TRSs and narrowing;

(iii) examples of using FRSs 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 FRSs. Some details of the theory of FRSs and their application to problem-oriented verification have been considered in [1, 3, 5]. In particular, the automated verification of several programs of text editing, array sorting and file sorting has been described.

References

[1] I. 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).

[2] I. Anureev, *Formula rewriting systems*, Preprint, Institute of Informatics Systems, Novosibirsk, Russia, 1997, No 40 (in Russian).

[3] I. Anureev, *Simplification procedures based on formula rewriting systems and intended for data types*, Preprint, Institute of Informatics Systems, Novosibirsk, Russia, 1998, No 54 (in Russian).

[4] I. Anureev, *Formula rewriting system theory*, Preprint, Institute of Informatics Systems, Novosibirsk, Russia, 1997, No 55 (in Russian).

[5] I. Anureev, *A method for simplification procedure design based on formula rewriting system* // Joint Bulletin of NCC&IIS. Ser.: Comput. Science, №8, 1998.

[6] V.A. Nepomniaschy, A.A. Sulimov *Problem-oriented knowledge bases and their application in program verification system SPECTRUM*, News of Russian Academy of Sciences. Theory and management systems, 1997, №2, 169—175 (in Russian).

[7] V.A. Nepomniaschy, A.A. Sulimov, *Problem-oriented means of program specification and verification in project SPECTRUM*, Lect. Notes Comput. Sci., **722**, 1993, 374–378.