Joint NCC & IIS Bull., Comp. Science, 15 (2001), 1–16
© 2001 NCC Publisher

Program verification based on the specification language SIMPLE*

I. S. Anureev

A new specification language, simple to master, is suggested. In spite of simplicity, this language is expressive enough. A decision method for its quantifier-free formulas is given. A program verification method based on this language is illustrated by an example of a program of bubble sorting.

Introduction

The practical use of modern verification systems is difficult due to complexity of specification languages describing the program properties.

A specification language of a typical program verification system [1–5] has a large number of heterogeneous syntactical structures, semantics (which is difficult for understanding and requires a profound knowledge in logic), a wide spectrum of methods and techniques of automatic proving, and numerous libraries of decision procedures for specific theories.

Mastering the variety of facilities of the specification language takes much time. Only then the facilities, which are most appropriate for verification of a specific class of programs, can be chosen. They usually form a small part of the arsenal of facilities. Time spent in studying other ones proves to be a waste time.

A specification language SIMPLE is represented in this work. This language can be mastered in one or two days.

A program verification method based on SIMPLE does not offer an alternative to the present verification systems. It is a filter of programs which are verified. We use sophisticated verification systems only when this method fails.

In spite of simplicity, SIMPLE is not a toy language. A class of programs which are verified with its help is representative. This language is a tool that improves efficiency of program verification.

*Partially supported by the Russian Foundation for Basic Research under Grants 00-01-00909 and 01-01-06141 and SB RAS Grant for Young Scientists.

1. Specification language SIMPLE

SIMPLE is a language of a first-order logic with two sorts "element" and "set" and a single predicate symbol "belong to".

1.1. Signature

A signature of SIMPLE is a quadruple $(F, \in, S, X)$, where F is a set of functional symbols, $\in$ is a predicate symbol "belong to", $S = \{Elem, Set\}$ is a set of sorts, X is a set of variables.

For every functional symbol, there is a nonnegative integer denoting its arity.

The set F is divided into two disjoint sets of determinate (F_d) and nondeterminate (F_n) functional symbols. The set F_d includes an indefinite value ω.

1.2. Terms

A set of terms T is built in the regular way:

- if $x \in X$, then x is a term;
- if $t_1, \ldots, t_n$ are terms, $f \in F$, then $f(t_1, \ldots, t_n)$ is a term.

The terms are divided into determinate (T_d) and nondeterminate (T_n) ones. A term is nondeterminate if it contains a nondeterminate functional symbol, otherwise it is determinate.

1.3. Formulas

The atomic formulas of SIMPLE look like $t \in s$, where $t \in T_d$ and $s \in T$.

The formulas of SIMPLE are built from atomic formulas with the help of propositional connectives and quantifiers:

- if A is an atomic formula, then A is a formula;
- $true$, $false$ are formulas;
- if A, B are formulas, then $\neg(A)$, $A \wedge B$, $A \vee B$, $A \Rightarrow B$, $A \Leftrightarrow B$ are formulas;
- if $x \in X$, A is a formula, then $\forall x(A)$, $\exists x(A)$ are formulas.

Let UF be a set of all quantifier-free formulas of SIMPLE, FS denote a set of all finite sets of UF, and FSS denote a set of all finite subsets of FS.

1.4. Semantics of signature

Define an interpretation I of symbols of the signature of SIMPLE:

- $I(Elem)$ is a set of elements;
- $I(Set)$ is a set of all subsets of the set $I(Elem)$;
- $I(\in)$ is a relation of membership;
- if $f \in F_d$, m is the arity of f, then $I(f)$ is a total function from $(I(Elem))^m$ to $I(Elem)$;
- if $f \in F_n$, m is the arity of f, then $I(f)$ is a total function from $(I(Elem))^m$ to $I(Set)$.

A function ϕ_I from X to $I(Elem)$ is called a meaning of variables in interpretation I.

1.5. Semantics of terms

A meaning ϕ_I extends to terms in the following way:
– if $t_1, \ldots, t_n \in T$, $f \in F_d$, then

$$\phi_I(f(t_1, \ldots, t_n)) = \{I(f)(a_1, \ldots, a_n) | a_1 \in \phi_I(t_1), \ldots, a_n \in \phi_I(t_n)\};$$

– if $t_1, \ldots, t_n \in T$, $f \in F_u$, then

$$\phi_I(f(t_1, \ldots, t_n)) = \bigcup_{a_1 \in \phi_I(t_1), \ldots, a_n \in \phi_I(t_n)} I(f)(a_1, \ldots, a_n).$$

A special kind of a meaning for determinate terms (a determinate meaning ϕ_I^d) is defined as follows:
– if $x \in X$, then $\phi_I^d(x) = \phi_I(x)$;
– if $t_1, \ldots, t_n \in T$, $f \in F_d$, then

$$\phi_I^d(f(t_1, \ldots, t_n)) = I(f)(\phi_I^d(t_1), \ldots, \phi_I^d(t_n)).$$

1.6. Semantics of formulas

Let I be an interpretation and ϕ_I be a meaning in I.

The semantics of propositional connectives and quantifiers is defined in an ordinary way. Therefore, it is sufficient to define semantics of atomic formulas.

An atomic formula $t \in s$ is valid in I and ϕ_I, if $\phi_I^d(t) \in \phi_I(s)$.

A formula A is valid in I, if A is valid in I and ϕ_I for each meaning ϕ_I. A formula A is valid, if A is valid in I for each interpretation I. A formula A is satisfiable in I, if A is valid in I and ϕ_I for some meaning ϕ_I.

An equality relation is expressed with the help of a predicate symbol $\in$. If $s \in T_d$, then $t \in s$ means that the terms t and s are equal.

Let M, N be sets with satisfiability relations, $\to$ be a relation on $M \times N$, and I be an interpretation. A relation $\to$ is said to preserve satisfiability, if for all $m \in M$, $n \in N$ such that $m \to n$ we have that m is satisfiable in I iff n is satisfiable in I.

Satisfiability extends to FS and FSS. A set $u \in FS$ is satisfiable in I, if $\bigwedge\limits_{A \in u} A$ is satisfiable in I. A set $U \in FSS$ is satisfiable in I, if $\bigvee\limits_{u \in U} \bigwedge\limits_{A \in u} A$ is satisfiable in I.

2. Formula completion rules

Formula completion rules are intended for completion of the sets of formulas. If the set contains a formula and its negation after completion, then it is unsatisfiable. These rules form the basis of a decision method for quantifier-free formulas of SIMPLE. The way of completion is given by a completion relation generated by a formula completion rule. Formula completion rules of four kinds are defined below. Each next kind of the rules includes all the previous ones. A criterion of satisfiability preservation for the sets of formulas is given for each kind of the rules.

For $u \in FS$, $Var(u)$ denotes a set of variables occurring in u. For $u_1, \ldots, u_n \in FS$, $Var(u_1, \ldots, u_n) = \bigcup\limits_{1 \le i \le n} Var(u_i)$.

2.1. Simple formula completion rules

A simple formula completion rule has the form $L \to R$, where $L, R \in FS$, and $Var(R) \subseteq Var(L)$.

Let ρ be a formula completion rule.

A completion relation $\to_\rho$ generated by ρ is defined as a set of pairs

$$(u, u \cup \{r\sigma \mid r \in R\}),$$

where $u \in FS$, σ is a substitution such that $\forall l \in L(l\sigma \in u)$.

Proposition 1. *Let the formula*

$$\bigwedge_{l \in L} l \Rightarrow \bigwedge_{r \in R} r$$

be valid in an interpretation I. Then $\to_\rho$ preserves satisfiability in I.

Proof. We have:

$$\rho \text{ has the form } L \to R; \tag{1}$$
$$u \to_\rho v; \tag{2}$$
$$\bigwedge_{l \in L} l \Rightarrow \bigwedge_{r \in R} r \text{ is valid in } I. \tag{3}$$

Then

$$u = u' \cup \{l\sigma | l \in L\}; \tag{4}$$
$$v = u' \cup \{l\sigma | l \in L\} \cup \{r\sigma | r \in R\} \text{ (by 1, 2)}. \tag{5}$$

The set u is satisfiable in I iff (by 4) $\bigwedge_{A \in u'} A \wedge \bigwedge_{l \in L} l\sigma$ is satisfiable in I iff (by 3) $\bigwedge_{A \in u'} A \wedge \bigwedge_{l \in L} l\sigma \wedge \bigwedge_{r \in R} r\sigma$ is satisfiable in I iff (by 5)

$$v \text{ is satisfiable in } I. \tag{6}$$

The relation $\to_\rho$ preserves satisfiability in I (by 6). □

2.2. Formula completion rules with case analysis

Formula completion rules with case analysis has the form

$$L \to R_1, \ldots, R_n,$$

where $L, R_1, \ldots, R_n \in FS$, and $Var(R_1, \ldots, R_n) \subseteq Var(L)$.

Let ρ be a formula completion rule with case analysis.

A completion relation $\to_\rho$ generated by ρ is defined as a set of pairs

$$(u, \{u \cup \{r\sigma \mid r \in R_1\}, \ldots, u \cup \{r\sigma \mid r \in R_n\}\}),$$

where $u \in FS$, σ is a substitution such that $\forall l \in L(l\sigma \in u)$.

Proposition 2. *Let the formula*

$$\bigwedge_{l \in L} l \Rightarrow \bigvee_{1 \le i \le n} \bigwedge_{r \in R_i} r$$

be valid in an interpretation I. Then $\to_\rho$ preserves satisfiability in I.

Proof. We have:

$$\rho \text{ has the form } L \to R_1, \ldots, R_n; \tag{1}$$
$$u \to_\rho V; \tag{2}$$
$$\bigwedge_{l \in L} l \Rightarrow \bigvee_{1 \le i \le n} \bigwedge_{r \in R_i} r \text{ is valid in } I. \tag{3}$$

Then

$$u = u' \cup \{l\sigma | l \in L\}; \tag{4}$$
$$V = \{u' \cup \{l\sigma | l \in L\} \cup \{r\sigma | r \in R_i\} | 1 \leq i \leq n\} \text{ (by 1, 2).} \tag{5}$$

The set u is satisfiable in I iff (by 4) $\bigwedge_{A \in u'} A \wedge \bigwedge_{l \in L} l\sigma$ is satisfiable in I iff (by 3) $\bigvee_{1 \leq i \leq n} (\bigwedge_{A \in u'} A \wedge \bigwedge_{l \in L} l\sigma \wedge \bigwedge_{r \in R_i} r\sigma)$ is satisfiable in I iff (by 5)

$$V \text{ is satisfiable in } I. \tag{6}$$

The relation $\rightarrow_\rho$ preserves satisfiability in I (by 6). □

2.3. Conditional formula completion rules

A conditional formula completion rule has the form

$$P_1, \ldots, P_n | L \rightarrow R_1, \ldots, R_n,$$

where $P_1, \ldots, P_n, L, R_1, \ldots, R_n \in FS$, and $Var(P_1, \ldots, P_n, R_1, \ldots, R_n) \subseteq Var(L)$.

Let ρ be a conditional formula completion rule.

A completion relation $\rightarrow_\rho$ generated by ρ is defined as a set of pairs

$$(u, \{u \cup \{p\sigma \mid p \in P_i\} \cup \{r\sigma \mid r \in R_j\} | 1 \leq i \leq n \wedge j \in J_i\}),$$

where $u \in FS$, σ is a substitution such that $l\sigma \in u$ for each $l \in L$, and $J_i = \{j | P_j = P_i \wedge 1 \leq j \leq n\}$.

Proposition 3. *Let the formulas*

$$\bigwedge_{l \in L} l \Rightarrow \bigvee_{1 \leq i \leq n} \bigwedge_{p_i \in P_i} p_i;$$
$$\bigwedge_{1 \leq i \leq n} (\bigwedge_{l \in L} l \wedge \bigwedge_{p \in P_i} p \Rightarrow \bigvee_{j \in J_i} \bigwedge_{r \in R_j} r)$$

be valid in an interpretation I. Then $\rightarrow_\rho$ preserves satisfiability in I.

Proof. We have:

$$\rho \text{ has the form } P_1, \ldots, P_n | L \rightarrow R_1, \ldots, R_n; \tag{1}$$
$$u \rightarrow_\rho V; \tag{2}$$
$$\bigwedge_{l \in L} l \Rightarrow \bigvee_{1 \leq i \leq n} \bigwedge_{p \in P_i} p; \tag{3}$$
$$\bigwedge_{1 \leq i \leq n} (\bigwedge_{l \in L} l \wedge \bigwedge_{p \in P_i} p \Rightarrow \bigvee_{j \in J_i} \bigwedge_{r \in R_j} r). \tag{4}$$

Then

$$u = u' \cup \{l\sigma | l \in L\}; \tag{5}$$
$$V = \{u' \cup \{l\sigma | l \in L\} \cup \{p\sigma | p \in P_i\} \cup \{r\sigma | r \in R_i\} | 1 \leq i \leq n\} \text{ (by 1, 2)}. \tag{6}$$

The set u is satisfiable in I iff (by 5) $\bigwedge_{A \in u'} A \wedge \bigwedge_{l \in L} l\sigma$ is satisfiable in I iff (by 3) $\bigvee_{1 \leq i \leq n} (\bigwedge_{A \in u'} A \wedge \bigwedge_{l \in L} l\sigma \bigwedge_{p \in P_i} p\sigma)$ is satisfiable in I iff (by 4) $\bigvee_{1 \leq i \leq n} \bigvee_{j \in J_i} (\bigwedge_{A \in u'} A \wedge \bigwedge_{l \in L} l\sigma \wedge \bigwedge_{p \in P_i} p\sigma \wedge \bigwedge_{r \in R_j} r\sigma)$ is satisfiable in I iff (by 6)

$$V \text{ is satisfiable in } I. \tag{7}$$

The relation $\rightarrow_\rho$ preserves satisfiability in I (by 7). □

2.4. Formula completion rules with extra variables

A formula completion rule with extra variables has the form

$$P_1, \ldots, P_n | L \rightarrow R_1, \ldots, R_n,$$

where $P_1, \ldots, P_n, L, R_1, \ldots, R_n \in FS$.

A restriction $Var(P_1, \ldots, P_n, R_1, \ldots, R_n) \subseteq Var(L)$ is removed. The formulas appearing in $P_1, \ldots, P_n, R_1, \ldots, R_n$ can contain extra variables.

Let ρ be a formula completion rule with extra variables.

A completion relation $\rightarrow_\rho$ generated by ρ is defined as a set of pairs

$$(u, \{u \cup \{p\sigma \mid p \in P_i\} \cup \{r\sigma \mid r \in R_j\} | 1 \leq i \leq n \wedge j \in J_i\}),$$

where $u \in FS$, σ is a substitution such that $\forall l \in L(l\sigma \in u)$, $Var(u) \cap Var(\rho) = \emptyset$, $J_i = \{j | P_j = P_i \wedge 1 \leq j \leq n\}$.

Proposition 4. *Let the formulas*

$$\bigwedge_{l \in L} l \Rightarrow \bigvee_{1 \leq i \leq n} \exists \bar{x}_i (\bigwedge_{p \in P_i} p) \textit{ where } \bar{x}_i = Var(P_i) \setminus Var(L);$$
$$\bigwedge_{1 \leq i \leq n} (\bigwedge_{l \in L} l \wedge \bigwedge_{p \in P_i} p \Rightarrow \bigvee_{j \in J_i} \exists \bar{y}_{ij} (\bigwedge_{r \in R_j} r)$$
$$\textit{where } \bar{y}_{ij} = Var(R_j) \setminus (Var(L) \cup Var(P_i)))$$

be valid in an interpretation I. Then $\rightarrow_\rho$ preserves satisfiability in I.

Proof. We have:

$$\rho \text{ has the form } P_1, \ldots, P_n | L \rightarrow R_1, \ldots, R_n; \tag{1}$$
$$u \rightarrow_\rho V; \tag{2}$$
$$\bigwedge_{l \in L} l \Rightarrow \bigvee_{1 \leq i \leq n} \exists \bar{x}_i (\bigwedge_{p \in P_i} p); \tag{3}$$
$$\bigwedge_{1 \leq i \leq n} (\bigwedge_{l \in L} l \wedge \bigwedge_{p \in P_i} p \Rightarrow \bigvee_{j \in J_i} \exists \bar{y}_{ij} (\bigwedge_{r \in R_j} r)). \tag{4}$$

Then

$$u = u' \cup \{l\sigma | l \in L\}; \tag{5}$$
$$V = \{u' \cup \{l\sigma | l \in L\} \cup \{p\sigma | p \in P_i\} \cup \{r\sigma | r \in R_i\} | 1 \leq i \leq n\} \text{ (by 1, 2)}. \tag{6}$$

The set u is satisfiable in I iff (by 5) $\bigwedge\limits_{A \in u'} A \wedge \bigwedge\limits_{l \in L} l\sigma$ is satisfiable in I iff (by 3) $\bigvee\limits_{1 \leq i \leq n} (\bigwedge\limits_{A \in u'} A \wedge \bigwedge\limits_{l \in L} l\sigma \bigwedge\limits_{p \in P_i} p\sigma)$ is satisfiable in I iff (by 4) $\bigvee\limits_{1 \leq i \leq n} \bigvee\limits_{j \in J_i} (\bigwedge\limits_{A \in u'} A \wedge \bigwedge\limits_{l \in L} l\sigma \wedge \bigwedge\limits_{p \in P_i} p\sigma \wedge \bigwedge\limits_{r \in R_j} r\sigma)$ is satisfiable in I iff (by 6)

$$V \text{ is satisfiable in } I. \tag{7}$$

The relation $\rightarrow_\rho$ preserves satisfiability in I (by 7). □

2.5. Formula completion systems

Formula completion rules with extra variables are the most general kind of formula completion rules. Later on by formula completion rules we mean formula completion rules with extra variables.

Formula completion rules extend to sets of sets of formulas.

A completion relation $\rightarrow_\rho$ generated by ρ is defined as a set of pairs $U \rightarrow_\rho U \setminus \{u\} \cup W$, where $u \in U$, $u \rightarrow_\rho W$.

A formula completion system is a finite set of formula completion rules.

A completion relation $\rightarrow_R$ generated a formula completion system R is defined as a union of completion relations generated by all formula completion rules of R.

Let I be an interpretation.

A formula completion rule ρ is correct in I if the formulas

$$\bigwedge_{l \in L} l \Rightarrow \bigvee_{1 \leq i \leq n} \exists \bar{x}_i (\bigwedge_{p \in P_i} p) \text{ where } \bar{x}_i = Var(P_i) \setminus Var(L);$$
$$\bigwedge_{1 \leq i \leq n} (\bigwedge_{l \in L} l \wedge \bigwedge_{p \in P_i} p \Rightarrow \bigvee_{j \in J_i} \exists \bar{y}_{ij} (\bigwedge_{r \in R_j} r)$$
$$\text{where } \bar{y}_{ij} = Var(R_j) \setminus (Var(L) \cup Var(P_i)))$$

are valid in I.

A formula completion system R is correct in I if all rules of R are correct in I.

Proposition 5. *Let a formula completion system R be correct in an interpretation I. Then $\rightarrow_R$ preserves satisfiability in I.*

Proof. By proposition 4, $\rightarrow_\rho$ preserves satisfiability for each $\rho \in R$. Then $\rightarrow_R = \bigcup\limits_{\rho \in R} \rightarrow_\rho$ preserves satisfiability. □

3. Decision method for quantifier-free formulas of SIMPLE

The decision method for quantifier-free formulas of SIMPLE is based on a special (normal) presentation of literals in a disjunctive normal form and on application of formula completion rules of a special (normal) kind to the disjunctive normal form with normal literals.

3.1. Main definitions

An atomic formula or its negation is called a literal.

A literal of the form

$$x \in f(x_1, \ldots, x_n) \text{ or } \neg(x \in f(x_1, \ldots, x_n)),$$

where $x, x_1, ..., x_n \in X$, and $f \in F$ is called a normal literal.

A set $U \in FSS$ is called a disjunctive normal form (DNF for short), if $\forall u \in U \forall A \in u$ (A is a literal).

A set $U \in FSS$ is called a disjunctive normal form with normal literals (NLDNF for short), if $\forall u \in U \forall A \in u$ (A is a normal literal).

A formula completion rule is normal, if all formulas which occur in it are normal literals. A formula completion system is normal if all rules of this system are normal.

3.2. Algorithm of reduction to NLDNF

Let $v \in FS$ and $V \in FSS$.

An algorithm of reduction of a formula $A \in UF$ to NLDNF consists of three steps:

Step 1. A is reduced to the form $A' = \bigvee\limits_{i \in I} (\bigwedge\limits_{j \in J} A_{ij})$, where A_{ij} are literals, by usual logical transformations;

Step 2. A' is rewritten to $U_A = \{\{A_{ij} | j \in J\} | i \in I\}$;

Step 3. U_A is rewritten to D_A, where D_A is a result of application of the following rules to U_A:

- $\{v \cup \{s \in f(t_1, \ldots, t_{i-1}, t_i, t_{i+1}, \ldots, t_n)\}\} \cup V$, where $t_i \notin X$ is replaced by $\{v \cup \{y \in t_i, s \in f(t_1, \ldots, t_{i-1}, y, t_{i+1}, \ldots, t_n)\}\} \cup V$, where $y \in X \setminus Var(u)$;
- $\{v \cup \{s \in t\}\} \cup V$, where $s \notin X$ is replaced by $\{v \cup \{y \in s, y \in t\}\} \cup V$, where $y \in X \setminus Var(u)$.

Proposition 6. *The algorithm of reduction to NLDNF terminates. Its result D_A is an NLDNF of A. The result D_A is satisfiable in I iff A is satisfiable in I for all interpretations I.*

Proof. Let us prove that D_A is an NLDNF of A.

A set D_A is DNF, since U_A is DNF and the rules of step 3 only transform literals into literals.

Suppose that $D_A = \{\{B\} \cup u\} \cup U$, where B is not a normal literal. Then B has the form $s \in f(t_1, \ldots, t_{i-1}, t_i, t_{i+1}, \ldots, t_n)$, where $t_i \notin X$, or $s \in t$, where $s \notin X$.

In the former case, the first rule of step 3 is applicable. In the latter case, the second rule of step 3 is applicable. This contradicts nonapplicability of the rules of step 3 to D_A.

Let us prove termination of the algorithm. Termination of steps 1 and 2 is evident.

Let $size(A)$ denote the number of functional symbols in A, $\prec'$ be a partial order on literals, such that $A \prec B'$ iff $size(A) < size(B)$, $\prec$ be a multiset extension of a multiset extension of $\prec'$.

For all $U, V \in FSS$, if U is rewritten by the rules of step 3 into V, then $V \prec U$.

Step 3 terminates, since $\prec$ is a well-founded relation.

Let I be an interpretation. Let us prove that D_A is satisfiable in I iff A is satisfiable in I. Step 1 preserves satisfiability, since the equivalent logical transformation is only used. Step 2 preserves satisfiability by definition of the satisfiability relation on FSS. It is sufficient to prove that both rules of step 3 preserve satisfiability.

Let U denote $\{v \cup \{s \in f(t_1, \ldots, t_{i-1}, t_i, t_{i+1}, \ldots, t_n)\}\} \cup V$ and U' denote $\{v \cup \{y \in t_i, s \in f(t_1, \ldots, t_{i-1}, y, t_{i+1}, \ldots, t_n)\}\} \cup V$.

Let U be satisfiable in I. Then U is valid in I and some ϕ_I. Let ϕ'_I be a meaning such that $\forall x \in Var(U)(\phi'_I(x) = \phi_I(x))$ and $\phi'_I(y) = \phi_I(t_i)$. Then U' is valid in I and ϕ'_I. So, U' is satisfiable in I.

Let U' be satisfiable in I. Then U' is valid in I and some ϕ_I. Thus, U is valid in I and ϕ_I. Hence U is satisfiable in I.

The first rule of step 3 preserves satisfiability.

Let U denote $\{v \cup \{s \in t\}\} \cup V$, and U' denote $\{v \cup \{y \in s, y \in t\}\} \cup V$.

Let U be satisfiable in I. Then U is valid in I and some ϕ_I. Let ϕ'_I be a meaning such that $\forall x \in Var(U)(\phi'_I(x) = \phi_I(x))$ and $\phi'_I(y) = \phi_I(s)$. Then U' is valid in I and ϕ'_I. Then U' is satisfiable in I.

Let U' be satisfiable in I. Then U' is valid in I and some ϕ_I. Then U is valid in I and ϕ_I. Hence U is satisfiable in I.

The second rule of step 3 preserves satisfiability. □

3.3. Decision algorithm for quantifier-free formulas of SIMPLE

Let A be an quantifier-free formula to be checked on satisfiability, I be an interpretation, R be an empty formula completion system (an empty set of formula completion rules).

The decision algorithm for quantifier-free formulas of SIMPLE (UFA for short) consists of the following steps:

Step 1. A is rewritten to V, where D_A is an NLDNF of A.

Step 2. R is extended with new normal formula completion rules which are correct in I.

Step 3. D_A is rewritten to D'_A, where D'_A is a result of application of the rules given below to D_A in the following order:

1. $\{\{false\} \cup v\} \cup V$, where $V \neq \emptyset$ is replaced by V;
2. $\{\{false\} \cup v\}$ is replaced by $\{\{false\}\}$;
3. $\{\{true\} \cup v\} \cup V$, where $v \neq \emptyset$, is replaced by $\{v\} \cup V$;
4. $\{\{true\}\} \cup V$ is replaced by $\{\{true\}\}$;
5. $\{\{B, \neg(B)\} \cup v\} \cup V$ is replaced by $\{\{false\} \cup v\} \cup V$;
6. $\{\{x \in x\} \cup v\} \cup V$ is replaced by $\{v\} \cup V$;
7. If $\{x \in t, y \in t\} \subseteq u$, where $t \in F_d$, then $\{v\} \cup V$ is replaced by $\{v[x \leftarrow y]\} \cup V$;
8. $\{\{x \in y\} \cup v\} \cup V$, where $x, y \in X$, is replaced by $\{v[x \leftarrow y]\} \cup V$;
9. if $V \to_R W$, then V is replaced by W.

Step 4. If D'_A has the form $\{\{true\}\}$ or $\{\{false\}\}$, then the algorithm terminates with the result D'_A. Otherwise go to step 2.

In rules 7 and 8, $v[x \leftarrow y]$ denotes a result of replacement of all occurrences of x in the formulas of v by y.

The algorithm UFA allows us to check unsatisfiability of formulas. It can be also used as a simplification algorithm if the number of iterations of steps 2 and 3 is restricted. A set V is called the k-th simplification of a formula A, if V is a result of k iterations of steps 2 and 3.

Proposition 7. *Let V be the k-th simplification of a formula $A \in UF$, and I be an interpretation. Then V is satisfiable in I iff A is satisfiable in I.*

Proof. Let us prove that each step of UFA preserves satisfiability. Step 1 preserves satisfiability by proposition 6. Steps 2 and 4 preserve satisfiability, since they do not rewrite a formula. Rules 1-5 of step 3 preserve satisfiability by the properties of logical constants and propositional connectives. Rule 6 obviously preserves satisfiability. By $t \in T_d$, $\phi_I(x \in t \wedge y \in t) \Rightarrow \phi_I(x) = \phi_I(y)$ for each meaning ϕ_I. Then rule 7 of step 3 preserves satisfiability. Since $\phi_I(x \in y) \Rightarrow \phi_I(x) = \phi_I(y)$, rule 8 of step 3 preserves satisfiability.

Rule 9 of step 3 preserves satisfiability by proposition 5. □

4. Expressiveness of formula completion rules

Formula completion rules allow us to express extensively used techniques of automatic proving (axiom application, term rewriting, case analysis, variable replacement) in a uniform manner.

4.1. Axiom application

The axiom application is expressed by simple formula completion rules. Application of an axiom $A_1 \wedge \ldots \wedge A_n \Rightarrow A$ corresponds to application of the rule

$$\{A_1, \ldots, A_n\} \to \{A\}.$$

4.2. Term rewriting

A term rewriting rule $l \to r$ [6] is expressed by a formula completion rule $L \to R$, where $x \in X \setminus Var(l, r)$, $\{L\}$, $\{R\}$ are NLDNF of $x \in l$, $x \in r$, respectively.

The problem of cycling of associative commutative term rewriting rules [7] disappears after transition to formula completion rules. For example, application of the term rewriting rule $x+y \to y+x$ results in cycling, whereas the corresponding formula completion rule $\{z \in x + y\} \to \{z \in y + x\}$ does not.

An analogue of a conditional term rewriting rule $A|l \to r$, where A is a conjunction of literals, has the form $P|L \to R$, where $x \in X \setminus Var(l, r)$, and $\{L\}$, $\{R\}$, $\{P\}$ are NLDNFs of $x \in l$, $x \in r$, A, respectively.

4.3. Case analysis

The case analysis is represented by formula completion rules with case analysis. The case analysis $A_1 \vee \ldots \vee A_n$ fulfilled on the assumption of A corresponds to application of a formula completion rule

$$\{A\} \to \{A_1\}, \ldots, \{A_n\}.$$

4.4. Variable replacement

A variable replacement is represented by formula completion rules with extra variables. A variable replacement $x_1 \to t_1, \dots, x_n \to t_n$ fulfilled on the assumption of A corresponds to application of a formula completion rule

$$\{x_1 \in t_1, \dots, x_n \in t_n\}|\{A\} \to \{true\}.$$

5. Program verification method based on SIMPLE

A program verification method based on SIMPLE is illustrated with a program for bubble sorting of an array.

5.1. Tools of program annotation

To annotate the program for bubble sorting, the following notions are introduced in SIMPLE.

1. Notions for arrays.

   Let a be an array and n, m be integers.

   – $a[n, m]$ is a tuple of elements of a from n-th to m-th. A result is an empty tuple if $n > m$;
   – $a[n]$ is a n-th element of a. If $n < 0$ then $a[n] = \omega$.

2. Notions $<=$, $<$, $>=$, $>$, $+$, $-$, 0, 1, 2, ... are defined for integers.

   Let n, m be integers.

   – $<= (n)$ is a set of integers which are less than or equal to n;
   – $< (n)$ is a set of integers which are less than n;
   – $>=$ and $>$ are defined in a similar manner;
   – $n + m$ and $n - m$ are the sum and remainder of n and m, correspondingly.

3. Notions *perm*, *ord*, $+$ and $<=$ are defined for tuples.

   Let x be a tuple.

   – $perm(x) = \{y | y \text{ is a permutation of } x\}$;
   – $ord = \{y | y \text{ is an ordered tuple}\}$;
   – $+$ is a concatenation of two tuples;
   – $<= (x) = \{y | \forall a \in y \forall b \in x (a \in <= (b))\}$.

5.2. Annotated program

An annotated program has the form

$pred$
$\mathbf{for(int\ i = 2; i <= n; i++)}$
$inv1$ $\mathbf{for(int\ j = n; j >= i; i--)}$
$inv2$ $\mathbf{if(a[j-1] > a[j])\{x = a[j-1]; a[j-1] = a[j]; a[j] = x\}}$
$post$

The code of the C language is in thick print. The formulas $pred$, $post$, $inv1$, $inv2$ of SIMPLE give a precondition, a postcondition, an invariant of an outer loop and an invariant of an inner loop, correspondingly.

The precondition $pred$ has the form $a[1,n] \in b \wedge n \in >= (1)$.

The postcondition $post$ has the form $a[1,n] \in perm(b) \wedge a[1,n] \in ord$.

The invariant $inv1$ has the form

$$n \in >= (1) \wedge a[1,n] \in perm(b) \wedge a[1,i-1] \in ord \wedge$$
$$a[1,i-2] \in <= (a[i,n]) \wedge i \in >= (2) \wedge i \in <= (n+1)).$$

The invariant $inv2$ has the form

$$inv1 \wedge a[j,j] \in <= (a[j+1,n]) \wedge j \in <= (n) \wedge j \in >= (i-1).$$

5.3. Verification conditions

Six verification conditions are generated by Floyd-Hoare method:

- $pred \Rightarrow (inv1[i \leftarrow 2])$;
- $inv1 \wedge i \in <= (n) \Rightarrow (inv2[j \leftarrow n])$;
- $inv1 \wedge \neg(i \in <= (n)) \Rightarrow post$;
- $inv2 \wedge j \in >= (i) \wedge a[j-1] \in > (a[j]) \Rightarrow$
  $(inv2[a \leftarrow upd(upd(a, j-1, a[j]), j, a[j-1]), j \leftarrow j-1])$;
- $inv2 \wedge j \in >= (i) \wedge \neg(a[j-1] \in > (a[j])) \Rightarrow (inv2[j \leftarrow j-1])$;
- $inv2 \wedge \neg(j \in >= (i)) \Rightarrow (inv1[i \leftarrow i+1])$.

Let us notice that, if annotations are quantifier-free formulas, then verification conditions to be generated are also quantifier-free formulas.

5.4. Proof of verification conditions

A proof of verification conditions consists in application of the algorithm UFA to negations of verification conditions. Validity of a verification condition corresponds to unsatisfiability of its negation.

To apply the algorithm, it is necessary to give formula completion rules for notions occurring in verification conditions. Here is a number of formula completion rules for notions *ord* and <=:

- $\{x \in ord, y \in ord, x \in\leq (y), z \in +(x,y)\} \to \{z \in ord\}$;
- $\{x \in a[i,i], x \in ord\} \to \{true\}$;
- $\{x \in<= (y[m,n]), n \in< (m)\} \to \{true\}$.

Moreover, there is a number of formula completion rules for arithmetics, for instance, $\{i \in>= (i)\} \to \{true\}$. A combination of the algorithm UFA with a decision procedure for Presburger arithmetics allows us to eliminate these rules.

If unsatisfiability of all negations of the verification conditions are proved, the initial annotated program is correct.

5.5. Error localization

If, after the k-th application of steps 2 and 3 of the algorithm UFA, new formula completion rules are not added, then the k-th simplification of the initial formula (negation of a verification condition) restricts input data with a possible wrong execution of the program. Execution of the program on input data which do not satisfy the k-th simplification is correct. Representation of this restriction in the form of NLDNF facilitates test generation.

6. Conclusion

The following results are obtained in this paper:

- A new specification language SIMPLE, which is simple and expressive enough, is suggested.
- A new method of automatic proving called formula completion systems is developed. Formula completion systems combine and unify well-known techniques, such as axiom application, term rewriting, case analysis and variable replacement.
- A decision method for quantifier-free formulas of SIMPLE is given.
- A program verification method based on SIMPLE is described.

At present, in the framework of the program verification system SPECTRUM [8, 9], a new prover partially based on SIMPLE is being implemented.

References

[1] Gerhart S. L., Musser D. R., et al. An overview of AFFIRM: A specification and verification system // Proc. IFIP Congress 80. — IFIP Congress Ser. — 1980. — Vol. 8. — P. 343–347.

[2] Gordon M. J. C., Melham T. F. Introduction to HOL: A Theorem Proving Environment for Higher Order Logic. — Cambridge University Press, 1993.

[3] Boyer R. S., Moore J. S. A Computational Logic Handbook. — Academic Press, 1988.

[4] Chen J., Han J. A Review of EVES. — St. Lucia, 1993. — (Tech. Rep. / Dept. Comput. Sci., Univ. of Queensland; N 93-5).

[5] Owre S., Shankar N., Rushby J. M. User Guide for the PVS Specification and Verification System. — Computer Science Laboratory, SRI International, Menlo Park, CA, 1993.

[6] Klop J. W. Term rewriting systems // Handbook of Logic in Computer Science. — 1993. — Vol. 2. — P. 1–116.

[7] Dershowitz N., Jouannaud J.-P. Rewrite systems // Handbook of Theoretical Computer Science. — 1990. — Vol. B(6). — P. 243–320.

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

[9] Nepomniaschy V. A., Sulimov A. A. Problem-oriented verification system and its application to linear algebra programs // Theor. Comput. Sci. — 1993. — Vol. 119. — P. 173–185.