Application of minimization algorithm for finite acyclic automata in finding condition’s basis for program invariant search.

O. M. Maksymets

Finding invariants subtask such as finding conditions basis is solved by using a minimization algorithm for finite acyclic automata. Conditions are interpreted as acyclic automata with one finite state after an arrangement of conditions and composition of assignment operators. Linear algorithm complexity is shown. The algorithm is implemented constructing lexical tree in Java using ANTLR Parser Generator.

Introduction

Verification of programs based on Floyd-Hoare-Dijkstra’s inductive approval, using pre/postconditions and loop invariants, has considered a actual problem of research in the seventies, that lead to the development of many system of program verification. However, limited progress has been made in achieving the goal of mechanical verification of program’s properties because: theorem proving, needed to establish the validity of verification conditions, were not powerful enough; and the user had to manually annotate programs with loop invariants, because of the few existing tools for this purpose at that moment could not meet the user’s needs.

For life-critical applications it is still imperative to verify properties of programs [5]. With substantial progress in automated reasoning, several techniques for verification have emerged in the form of static analysis of programs (type checking, type inference, extended static checking, etc.), model checking as well as verifying properties of software and hardware using theorem proving techniques. This work solves the subtask of the automatically generating loop invariants problem. Specifying, the finding conditions basis on current state of the program subtask.

Basic concepts

Let A be U-Y program over memory with set of variables \( R = \{r_1, \ldots, r_m\} \) that defined on algebra of data \((D, \Omega)\). \( K(\Omega, Eq)\) is an algebra class that includes algebra \((D, \Omega)\) and is defined with the set of equalities \(Eq\) [1]. \( T(\Omega, R)\) is free algebra of terms on \( R \) from class \( K(\Omega, Eq)\). We consider absolutely free algebra, therefore \( Eq = \emptyset \).

We consider the problem of invariant search for language \( L \) that consists of condition defined as equalities \( r_i := h(r) \), where \( h(r) \in T(\Omega, R) \), \( r = (r_1, \ldots, r_m) \). We use tree structures for expression representation.

If \( M \) is a set of equalities then algebraic closure of \( M \) with respect to \( Eq \) called the smallest set \( C(M) \) that includes reflexive transitive symmetric closure of \( M \) and \( n \)-ary operation \( \omega \triangledown (g_1(r), q_1(r), \ldots, g_n(r), q_n(r)) \in C(M) \Rightarrow (\omega(g_1(r), \ldots, g_n(r)), \omega(q_1(r), \ldots, q_n(r))) \in C(M) \).

\( P \) is called algebraic basis of \( M \) if \( P \) is a minimal subset that \( C(P) = M \).

One of the main problems of finding invariants is to build basis of equalities for specific state of U-Y program. Using an iterative algorithm we have to compose conditions from a previous step with condition on a current step. If \( y \) are the conditions from previous step and \( y_1 \) is condition on a current step then \( y_2 = y_1 \ast y \) is advanced conditions on a current step and is a superposition of conditions \( y_1 \) and \( y \).

Algorithm of composition

Input: \( U \) is merged conditions \( y \) and \( y_1 \).
Output: advanced conditions \( y_2 \).
\( y_2 := \emptyset \)

for all conditions \( r_j := h \) in \( U \) do

for all \( r_k \) in \( h \) do

if \( r_k \) exists in \( y_2 \) then

replace \( r_k \) in \( h \) with expression of \( r_k \) in \( y_2 \)

end if

end for

if \( r_j \) exists in \( y_2 \) then

replace condition \( r_j := p \) in \( y_2 \) with condition \( r_j := h \)

else

\( y_2 := \text{merge}(y_2, r_j := h) \)

end if

end for

We suppose that operation \( \text{exist} \) in \( y_2 \) costs \( o(1) \). Nested loops proceed the walking through the \( U \). Therefore, the complexity of algorithm is \( o(n_U) \), where \( n_U \) is number of nodes in tree representation of \( U \).

A finite (outputless) automaton \( \Upsilon \) is the 4-tuple \((A, X, \delta, F)\), where \( A \) is the finite state set \((|A| = n_A)\), \( X \) is the finite alphabet of input symbols \((|X| = m_X)\), \( \delta : A \ast X \rightarrow A \) is the partial transition function, \( F \subset A \) is the set of finite states. If the initial state \( a_0 \) is distinguished in the set \( A \), then the automaton \( \Upsilon \) is called initialized \((\Upsilon(a_0))\).

Let \( X^* \) be the semigroup of all words in alphabet \( X \), including the empty word \( e \). We extended the transition function \( \delta \) to the entire set \( X^* \), setting

\[
(\forall a \in A) (\forall x \in X) (\forall p' \in X^*) (\delta(a, p) = \delta(a, x \cdot p')),
\]

where \( p = x \cdot p' \). If \( \delta(a, x) \) is undefined, then \( \delta(a, p) \) is also undefined.

The automation equivalence relation \( \alpha \) is defined as follows:

\[
aa a' \Leftrightarrow (\forall p \in X^*) (\delta(a, p) \in F \Leftrightarrow \delta(a', p) \in F).
\]

The construction of the factor automaton \( \Upsilon/\alpha \) is a classical problem in finite automata theory [1]. It is shown in [2] that if the automaton \( \Upsilon \) is acyclic (i.e., if \( p \in X^*, p \neq e \), \( \delta(a, p) = b \), then \( a \not\sim ab \), where \( a \not\sim ab \) reads that \( a \) and \( b \) are not automaton equivalent), then the minimization algorithm of [1] can be transformed into a minimization algorithm for finite acyclic automata with a linear upper bound on time complexity by the number of transitions in the original automaton. The proof of the algorithm complexity is stated in [2].

Let tree structure of advanced conditions \( y_2 \) interpret as acyclic automaton. \( A \) is a set of nodes in the tree and added finite state \( a^* \) which is connected with every leave of the tree. \( X = (\omega, i) \), where \( \omega \in \Omega \), \( i \) is a number of argument for \( \omega \) and is less or equal to arity of operation \( \omega \). \( r_i \) is presented as operation with zero arity. \( F = \{a^*\}, \delta : A \ast X \rightarrow A \) is defined by the tree edges.

Let define the initial partition of automaton’s states as \( R_{rank} = \{S_0, S_1, ..., S_k\} \), where

\[
S_0 = \{a \in A| (\forall x \in X) \delta(a, x) = \tau \};
\]

\[
S_{i+1} = \{a \in A| (\forall x \in X) ((\delta(a, x) = \tau \lor \delta(a, x) \in \bigcup_{j=1}^{i} S_j) \land (\exists x \in X) \delta(a, x) \in S_i)\};
\]

and \( \tau \) stands for undefined.

If \( L, L' \) are subsets of \( A \) then \( L < L' \Leftrightarrow L = \{a \in A| (\exists a' \in L') (\exists x \in X) f(a', x) = a \} \).
A minimization algorithm for finite acyclic automata

**Input:** automaton \( \Upsilon \) and partition \( R_{rank} \).

**Output:** partition \( R \) that presents classes of equal states.

\[
R := R_{rank} \\
\text{for all class } K \text{ in } R, \text{ in ascending order do} \\
\text{for all } x \text{ in } X \text{ do} \\
\quad \text{for all class } L \text{ in } R, \text{ that } L > K \text{ do} \\
\quad \quad L' := \{ a \in L \mid f(a, x) \in K \}; \\
\quad \quad L'' := \{ a \in L \mid f(a, x) \notin K \}; \\
\quad \quad \text{if } L' \neq \emptyset \text{ and } L'' \neq \emptyset \text{ then} \\
\quad \quad \quad \text{replace } L \text{ in } R \text{ with } \{L', L''\}; \\
\quad \text{end if} \\
\text{end for} \\
\text{end for} \\
\text{end for}
\]

Using partition \( R \) we decide identity of terms, matching of subterms and modify \( y_2 \). In this way we can guarantee that the set of conditions \( y_2 \) is minimal that represents the algebraic closure of conditions on current step.

**Conclusion**

We show how to find conditions basis and apply a minimization algorithm for finite acyclic automata in verification theory. The algorithm is implemented constructing lexical tree in Java using ANTLR Parser Generator. The developed Java class of conditions could be applied in implementation of verification algorithms in [3] and [4]. The linear basis finding algorithm complexity is argument for reasonable application.

**References**


**Authors**

Oleksandr Mykolayovych Maksymets — the 1st year postgraduate student, Department of Information Systems, Faculty of Cybernetics, Taras Shevchenko national university of Kiev, Kiev, Ukraine; E-mail: maksymets@gmail.com