Small program which demonstrates difficulties with deductive verification automation

Dmitry Kondratyev

This talk is devoted to the memory of outstanding scientist Valery Nepomniaschy

Deductive verification of big progamming systems

- Deductive verification of 23 unmodified Linux kernel library functions using using AstraVer toolset:

  Efremov D., Mandrykin M., Khoroshilov A. Deductive Verification of Unmodified Linux Kernel Library Functions. Lecture Notes in Computer Science. 2018. vol 11245. pp. 216–234. DOI: `https://doi.org/10.1007/978-3-030-03421-4_15`

- Deductive verification of JavaCard Virtual Machine implementation (over 7,000 lines of C code):

  Djoudi A., Hána M., Kosmatov N. Formal Verification of a JavaCard Virtual Machine with Frama-C. Lecture Notes in Computer Science. 2021. Volume 13047. pp. 427–444. DOI: `https://doi.org/10.1007/978-3-030-90870-6_23`

Task of automation of deductive verification

Challenges:

1. Problem of loop invariants
2. Problem of error localization
3. Problem of automatic verification condition proving

Solutions:

1. Symbolic method of verification of definite iterations
2. Semantic labeling method
3. Strategies for proving verification conditions

The C-lightVer system: overview

- Correct methods / algorithms at each step.
- Solution:

> *"Restrictions that contribute to provability are what make a programming language good."* Tony Hoare

- C-light language
  - covers the majority of C99 (C0 — completely, Misra C — almost);
  - sets the calculation order;
  - doesn't have some low-level operations.
- C-kernel language
  - is defined in terms of operational semantics;
  - axiomatic semantics is correct with respect to operational one.

Problem of loop invariants

Inference rule for *while* loop

$$\frac{\begin{array}{l}\{P\}\ \text{prog};\ \{I\},\\ \{I \wedge B\}\ \text{S}\ \{I\},\\ \qquad I \wedge \neg B \to Q\end{array}}{\{P\}\ \text{prog};\ \text{while B inv I do S}\ \{Q\}}$$

Solution for special case of loops:

Symbolic method of verification of definite iterations

Symbolic method of verification of definite iterations

Given

- $memb(S)$ denotes the multiset of elements of a structure S
- $empty(S) = true$ iff $|memb(S)| = 0$

let us define

1. $choo(S)$ returns an arbitrary element of $memb(S)$, if $\neg empty(S)$.
2. $rest(S) = S'$, where $memb(S') = memb(S) \setminus \{choo(S)\}$, if $\neg empty(S)$.

A definite iteration corresponds to the form:

$$\text{for x in S do v} := \text{body}(\text{v}, \text{x})$$

where

- S is a data structure
- x is a variable of type "element of S"
- v is a tuple of the loop variables excluding x
- $body$ represents the loop body which does not alter x and terminates for every $x \in S$

Replacement operation (*rep* function)

Let v_0 denote the initial values of variables from v.

Let us define replacement operation $rep(v, S, body)$ for this loop

1. if $empty(S)$, then $rep(v_0, S, body) = v_0$,
2. if $\neg empty(S)$, then
   $rep(v_0, S, body) = body(rep(v_0, rest(S), body), choo(S))$.

We suggest the following solution in the case of `break` statement presence in a definite iteration: when the loop exit is occurred by the execution of this statement, we assume that the loop iterations are continued, but the values of v remain unchanged.

Special Case of Definite Iteration

$$\text{for}\,(\mathrm{i}=0;\mathrm{i}<\mathrm{n};\mathrm{i}++)\ \mathrm{v}:=\mathrm{body}(\mathrm{v},\mathrm{i})\ \text{end},$$

where

- v — vector of modifying variables;
- a — one-dimensional array of n elements;
- $a \in v$;
- *body* — acceptable construction.

Iteration over Changeble Data Structures with Loop Exit

If $vec(a) = [a_1, a_2, \ldots, a_n]$ then let $rep(v, a, body, n)$:

- $rep(v, a, body, 0) = v_0$,
- $rep(v, a, body, i) =$
  $body(rep(v, a, body, i-1), a_{n-i})$
  for each $i = 1, 2, \ldots, n$.

If loop exit occured at iteration i $(0 < i \leq n)$, then for each j $(i \leq j \leq n)$:

$$rep(v, a, body, i) = rep(v, a, body, j)$$

Inference Rule:

$$\frac{\{P\}\,\mathrm{pr};\{Q(v \leftarrow rep(v, S, body, n))\}}{\{P\}\,\mathrm{pr};\ \mathrm{for}\,(\mathrm{i} = 0; \mathrm{i} < \mathrm{n}; \mathrm{i}++)\ \mathrm{v} := \mathrm{body}(\mathrm{v}, \mathrm{i})\ \mathrm{end}\{Q\}}$$

Problem of automatic verification condition proving

Automatic verification condition proving for programs with loops

Challenges:

- `break` statements in loops
- *rep* functions in verification conditions
- problems of proofs by induction using SMT-solvers

Solutions:

Use of ACL2 theorem prover

Strategies for proving verification conditions

ACL2 theorem prover

We use ACL2 as theorem prover in the C-lightVer system.

Applicative Common Lisp (ACL) is an input language of the ACL2 system.

The C-lightVer system generates verification conditions written in ACL language.

Obtained verification conditions with applications of recursive functions correspond to ACL2 logic based on computable recursive functions

Strategies for proving verification conditions

Goal

$$\psi \equiv \phi \to X,$$

where X contains $rep(n, \ \ldots)$

Two cases:

- ψ-*lemma*-1 : $\phi \to rep(n, \ldots).loop\text{-}break$
- ψ-*lemma*-2 : $\phi \to \neg rep(n, \ldots).loop\text{-}break$

Example

C-light program

```
1. /*@ requires (0 < n) && (n <= len(a));
2.     ensures (grt-eql-cnt(n, key, a) == 0 ==>
                                 \result == 0) &&
3.             (grt-eql-cnt(n, key, a) > 0 ==>
                                 \result == 1)
4. */
5. int grt_eql_key(int n, int key, int a[]){
6.    int i, result = 0;
7.    for (i = 0 ; i < n; i++){
8.        if (a[i] >= key){result = 1; break;}}
9.    return result;}
```

Example

vc-1

$$\begin{gathered}
\forall n, key, a \\
((0 < n \wedge n \leq len(a) \wedge n \in Int \wedge key \in Int \wedge \\
a \in IntArr \wedge grt\text{-}eql\text{-}cnt(n, key, a) = 0 \\
\rightarrow \\
rep(n, key, a, 0).result = 0) \\
\wedge \\
(0 < n \wedge n \leq len(a) \wedge n \in Int \wedge key \in Int \wedge \\
a \in IntArr \wedge grt\text{-}eql\text{-}cnt(n, key, a) > 0 \\
\rightarrow \\
rep(n, key, a, 0).result = 1))
\end{gathered}$$

Example

vc-1-*lemma*-1:

$$\begin{gathered}\forall n, key, a\\ (0 < n \wedge n \leq len(a) \wedge n \in Int \wedge key \in Int \wedge\\ a \in IntArr \wedge grt\text{-}eql\text{-}cnt(n, key, a) = 0\\ \rightarrow\\ \neg rep(n, key, a, 0).loop\text{-}break)\end{gathered}$$

vc-1-*lemma*-2:

$$\begin{gathered}\forall n, key, a\\ (0 < n \wedge n \leq len(a) \wedge n \in Int \wedge key \in Int \wedge\\ a \in IntArr \wedge grt\text{-}eql\text{-}cnt(n, key, a) > 0\\ \rightarrow\\ rep(n, key, a, 0).loop\text{-}break)\end{gathered}$$

ACL2 has proved verification condition by induction on n using these lemmas.

Conclusion

Towards automatic deductive verification of C programs:

1. Symbolic method of verification of definite iterations
2. Method of error localization
3. Strategies for proving verification conditions

Plans:

- Extensions of symbolic method of verification of definite iterations
- Strategies for error localization
- New strategies for proving verification conditions

References

- Nepomniaschy, V.A., Ryakin, O.M.: Applied methods of program verification. In: Radio and Communication, 256 p. Moscow (1988). (in Russian)
- Nepomniaschy V.A. Symbolic method of verification of definite iterations over altered data structures. Programming and Computer Software. 2005. Volume 31. Issue 1. pp. 1–9. DOI: https://doi.org/10.1007/s11086-005-0001-0
- Kondratyev D.A., Maryasov I.V., Nepomniaschy V.A. The Automation of C Program Verification by the Symbolic Method of Loop Invariant Elimination. Automatic Control and Computer Sciences. 2019. Volume 53. Issue 7. pp. 653–662. DOI: https://doi.org/10.3103/S0146411619070101
- Kondratyev D.A., Promsky A.V. The Complex Approach of the C-lightVer System to the Automated Error Localization in C-Programs. Automatic Control and Computer Sciences. 2020. Volume 54. Issue 7. pp. 728–739. DOI: https://doi.org/10.3103/S0146411620070093

Small program which demonstrates difficulties with deductive verification automation

Dmitry Kondratyev

This talk is devoted to the memory of outstanding scientist Valery Nepomniaschy