SOFTWARE VERIFICATION BASED ON TRANSFORMATION FROM PROCEDURAL PROGRAMS TO REWRITE SYSTEMS

Naoki Nishida, Yuki Furuichi, Masahiko Sakai, Keiichirou Kusakari, and Toshiki Sakabe

Graduate School of Information Science, Nagoya University

ABSTRACT

In our research, taking advantage of methods for proving *inductive theorems*, we apply them to verification of procedural programs written in a subset of the C language with integer type. More precisely, we transform procedural programs to equivalent rewrite systems, and verify that the rewrite systems satisfy the specifications, using the *inductionless induction* method. In this paper, we briefly summarize the outline of our approach.

1. INTRODUCTION

In the field of term rewriting, *inductionless induction* [10, 8] and *rewriting induction* [11, 3] have been widely studied as methods for proving *inductive theorems* [4]. Since equivalence of functions can be represented as inductive theorems, methods for proving inductive theorems are useful to verify the equivalence of functions in functional programming.

On the other hand, in the field of procedural programming, *Hoare logic* is useful in verifying correctness of functions written as 'while' programs [7, 9]. This method needs some heuristics for finding appropriate 'loop invariants' and 'pre- and post-conditions'.

In our research, taking advantage of methods for proving inductive theorems, we try to verify that procedural programs written as 'while' programs satisfy the corresponding specifications written as rewrite systems. More precisely, we propose a transformation from 'while' programs in a subset of C into rewrite systems and show that the transformation reduces the equivalence of procedural programs to that of functions in rewrite systems (cf. [6]). This paper briefly summarize this approach.

2. OUTLINE OF OUR APPROACH

Let $\mathcal{P}$ be a 'while' program in a subset of C [7], and $\mathcal{S}$ be a specification written as a *term rewriting system* (*TRS*) [1]. We first transform $\mathcal{P}$ and $\mathcal{S}$ into a rewrite system $\mathcal{R}$, and then apply the method of inductionless induction (more precisely, *completion* [1, 2]) to the pair of $\mathcal{R}$ and the equation e that represents equivalence between $\mathcal{P}$ and $\mathcal{S}$ (see Fig. 1). We show that if the completion process finishes successfully then $\mathcal{P}$ is proved to satisfy $\mathcal{S}$.

2.1. From Procedural to Functional

To apply techniques based on inductionless induction for abstract reduction systems, we transform 'while' programs with integer type into computationally equivalent *pdTRSs* whose rewrite rules have *Presburger sentences* [5] as their conditional parts. Since the Presburger sentences are decidable, it is easier to treat pdTRSs than conditional TRSs. We prove that the transformation is correct, i.e., computational equivalence between programs and the corresponding pdTRSs. This implies that computational properties of 'while' programs are reduced to that of 'functional' programs encoded by pdTRSs.

It is also possible to transform procedural programs into equivalent unconditional TRSs instead of pdTRSs. However, in our approach, we do not employ such a transformation. The reason is that encoding negative integers is easier on pdTRSs than on TRSs, and intermediate results of completion processes on pdTRSs are simpler than on TRSs.

2.2. Verification by Inductionless Induction

Inductionless induction [10, 8] is a method to show that two reduction relations are equivalent, and it is usable to prove that an equation e is an *inductive theorem* on a reduction system $\mathcal{R}$. In the process of the method, we do the following in order:

1. using a *completion* procedure, construct a *convergent* (i.e., *confluent* and *terminating*) rewrite system $\mathcal{R}'$ such that the reduction of $\mathcal{R}'$ is equivalent to that of the composition of $\{e\}$ and $\mathcal{R}$, and
2. check the condition that the set of all *normal forms* for $\mathcal{R}'$ is equal to that for $\mathcal{R}$.

If $\mathcal{R}$ is *sufficient-complete* and *confluent*, then the condition on normal forms at the above second item always holds. Since the transformation produces sufficient-complete and confluent pdTRSs, we can focus only on the above first item, i.e., 'completion' for the pair $(\{e\}, \mathcal{R})$.

In the field of term rewriting, completion procedures are well investigated for TRSs but not for pdTRSs. Therefore, we proposed a completion procedure for pdTRSs, based on the *KB completion* [1, 2]. Since TRSs are pdTRSs, the completion procedure we proposed is an extension of the KB completion procedure.

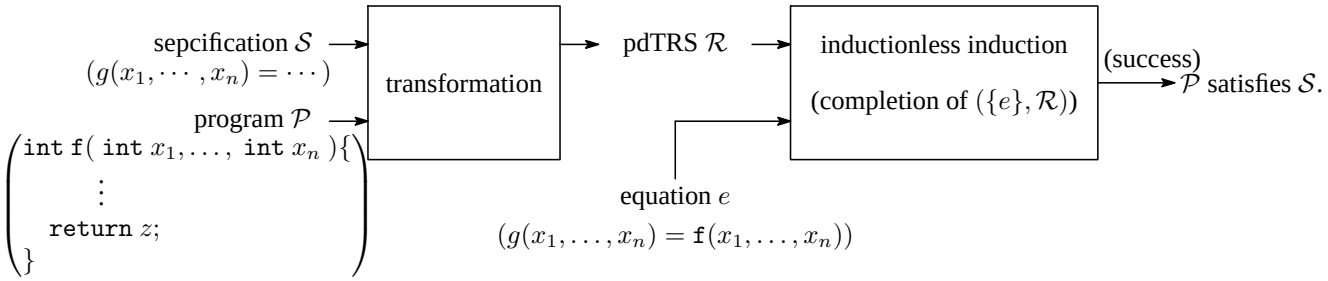

Fig. 1. outline of our approach

3. EXAMPLE OF VERIFICATION

In this section, we show an example of verification by our approach. Consider the following C program and specification:

```
int sum1( int x ){
   int i, z = 0;
    for( i = 0 ; i < x ; i++ ){
        z += i+1;
    }
    return z;
}
```

$$Sum(0) = 0 \text{ and } Sum(S(x)) = Sum(x) + S(x).$$

Here, natural numbers are encoded by function symbols S and 0. The above program and specification is transformed into the following pdTRS:

$$\mathcal{R} = \left\{ \begin{array}{l} \mathtt{sum1}(x) \to U_2(x, 0, 0) \\ U_2(x, i, z) \to U_2(x, S(i), z + S(i)) \quad \text{if } i < x \\ U_2(x, i, z) \to z \quad \text{if } i \geq x \\ Sum(0) \to 0 \\ Sum(S(x)) \to Sum(x) + S(x) \\ 0 + y \to y \\ S(x) + y \to S(x + y) \end{array} \right. .$$

To prove that `sum1` satisfies the specification, we apply the completion procedure to $(\{\mathtt{sum1}(x) = Sum(x)\}, \mathcal{R})$. We succeeded in the completion, by introducing the appropriate lemma "$U_2(S(x), i, y) = U_2(x, i, y) + S(x)$ if $i \leq x$". Therefore, it is guaranteed by our approach that `sum1` and Sum are equivalent computationally, i.e., the `sum1` program is correct as summation.

4. CONCLUSION

Although our approach is applicable to restricted C programs, it is expected that our approach is useful in checking C programs written by beginners at programming. Implementing a system for checking such C programs is one of our future works.

5. REFERENCES

[1] F. Baader and T. Nipkow, *Term Rewriting and All That*, Cambridge University Press, 1998.

[2] L. Bachmair, *Canonical Equational Proofs*, Birkhauser, 1991.

[3] A. Bouhoula, "Automated theorem proving by test set induction," *Journal of Symbolic Computation*, vol. 23, no. 1, pp. 47–77, 1997.

[4] R. S. Boyer and J. S. Moore, *A Computational Logic*, Academic Press, New York, 1979.

[5] D. C. Cooper, "Theorem proving in arithmetic without multiplication," *Machine Intelligence* 7, 1972, pp. 91–99, Edinburgh University Press.

[6] Y. Furuichi, N. Nishida, M. Sakai, K. Kusakari, and T. Sakabe, "Approach to software verification based on transformation from procedural programs to rewrite systems," IEICE Technical Report, 2006 (to appear, in Japanese).

[7] M. Hennessy, *The Semantics of Programming Languages: an Elementary Introduction using Structural Operational Semantics*, John Wiley & Sons, 1990.

[8] G. P. Huet and J.-M. Hullot, "Proofs by induction in equational theories with constructors," *Journal of Computer and System Sciences*, vol. 25, no. 2, pp. 239–266, 1982.

[9] M. Huth and M. Ryan, *Logic in Computer Science: Modelling and Reasoning about Systems*, Cambridge University Press, 2000.

[10] D. R. Musser, "On proving inductive properties of abstract data types," in *Conference Record of the Seventh Annual ACM Symposium on Principles of Programming Languages*, 1980, pp. 154–162.

[11] U. S. Reddy, "Term rewriting induction," in *Proceedings of the 10th International Conference on Automated Deduction*, 1990, vol. 449 of *Lecture Notes in Computer Science*, pp. 162–177, Springer.