ISSN 1726-9679

DANUBE ADRIA ASSOCIATION FOR AUTOMATION & MANUFACTURING

DAAAM INTERNATIONAL VIENNA

www.daaam.com

ANNALS OF DAAAM FOR 2011 &
PROCEEDINGS

OF THE 22ND INTERNATIONAL DAAAM SYMPOSIUM
"INTELLIGENT MANUFACTURING & AUTOMATION:
POWER OF KNOWLEDGE AND CREATIVITY"
23-26TH NOVEMBER 2011, VIENNA, AUSTRIA

ORGANIZED BY:
DAAAM INTERNATIONAL VIENNA
INTERNATIONAL ACADEMY OF ENGINEERING
VIENNA UNIVERSITY OF TECHNOLOGY, UNIVERSITY OF APPLIED SCIENCES
TECHNIKUM VIENNA AND AUSTRIAN SOCIETY OF ENGINEERS AND ARCHITECTS - ÖIAV 1848

UNDER THE AUSPICES OF: THE DANUBE RECTORS' CONFERENCE &
RECTORS' HONOR COMMITTEE OF DAAAM INTERNATIONAL

EDITOR: B.[RANKO] KATALINIC

Annals of DAAAM for 2011 & Proceedings of the 22nd International DAAAM Symposium, Volume 22, No. 1, ISSN 1726-9679
ISBN 978-3-901509-83-4, Editor B. Katalinic, Published by DAAAM International, Vienna, Austria, EU, 2011
Make Harmony between Technology and Nature, and Your Mind will Fly Free as a Bird
Annals & Proceedings of DAAAM International 2011

ALIGATOR: EXPERIMENTS AND LIMITATIONS

KOVACS, L[aura] & KOVACS, A[dalbert]

Abstract: *We address the problem of loop invariant generation of programs over scalars and unbounded data structures. Our approach to invariant generation is implemented in the Aligator software tool. We investigate various loop classes for using Aligator and presents experimental evaluation of our method. We also discuss limitations of the approach and underline the basic principles of how and when Aligator can be successfully applied.*
Key words: *program verification, loop invariants, polynomial relations, quantified first-order formulas*

1. INTRODUCTION

The complexity of computer systems grows exponentially. Many companies and organisations are now routinely dealing with software comprising several millions lines of code, written using different programming languages and styles. As a result, ensuring the reliability and safety of such software is extremely difficult. The main difficulty comes with the challenge of automatically deriving program properties from which safety of the software under analysis can be proved. One way of generating such properties is using computer-aided tools based on symbolic computation and computational logic.

In (Kovacs, 2008) an automated framework for the verification of program loops is introduced. The approach combines algorithmic combinatorics, computer algebra and automated reasoning, and aims at deriving valid polynomial loop properties, called loop invariants. The method is implemented in the Aligator software tool and was further extended in (Henzinger et al, 2010) to also generate quantified loop invariants for programs over unbounded data structures, such as arrays. The efficiency of Aligator depends and is restricted to special classes of programs. However, in practice, it turns out that the programming model of Aligator scales well on a large number of programs.

In this paper, we address the limitations and power of Aligator, with the purpose of answering what and what cannot be done with Aligator (Section 4). To this end, we report on experimental evaluations obtained by Aligator (Section 4.1), and formulate limitations of Aligator's programming model (Section 4.2). Our experiments give practical evidence for using Aligator in reasoning about loops. We thus believe that our method makes a significant step towards the automated verification of safety-critical systems.

2. RELATED WORK

Research into loop invariant generation has a long tradition. We only overview two lines of work that are the most related to Aligator, and refer to (Kovacs, 2008) for more details.

In (Rodriguez-Carbonell & Kapur, 2007) a method using *abstract interpretation and polynomial algebra* is presented to generate polynomial loop invariants for so-called simple loops. Unlike this work, Aligator handles more general loops and uses no abstract interpretation.

Abstract interpretation is also employed in (Gulwani et al, 2009), where *predicate templates* are used to generate quantified loop invariants over arrays. Contrarily to this approach, Aligator generates polynomial and quantified invariants without the burden of a priori specified predicates.

3. ALIGATOR: PRELIMINARIES

The programming model of Aligator is restricted to *P-solvable loops* (Kovacs, 2008). Informally, a loop is P-solvable if the following conditions are satisfied: (i) the loop body consists only of nested conditionals and assignments, (ii) assignments to scalars form a linear recurrence system with constant coefficients, (iii) closed forms of scalar variables can be represented by a polynomial system over loop counters and some extra variables where there are polynomial relations among the extra variables, and (iii) updates to arrays can be handled as uninterpreted functions over array and scalar variables. For such loops, (i) the polynomial invariant ideal over the scalar variables is derived by omitting loop tests and deploying recurrence solving algorithms, and (ii) quantified loop invariants over arrays are inferred by using loop tests in conjunction with the recurrence equations of scalars. The algoritmic approach of Aligator is in detail presented in (Kovacs, 2008) and (Henzinger et al, 2010).

Example 1. Consider the loop given in Figure 2(a). First, Aligator translates Figure 1(a) into Figure 1(b) by omitting test conditions. Next, recurrence solving and polynomial algebra techniques are deployed, and $xy - 1 = 0$ is derived as a polynomial loop invariant. That is, $xy - 1 = 0$ is a valid loop property before and after each loop iteration of Figure 1(b). Moreover, Aligator is complete. In other words, any other polynomial loop invariant of Figure 1(b) is proved to be a logical consequence of $xy - 1 = 0$. Hence, $xy - 1 = 0$ generates the polynomial invariant ideal of Figure 1(b).

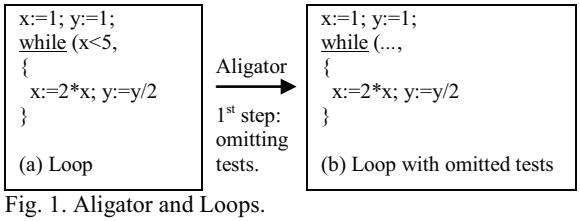

Fig. 1. Aligator and Loops.

4. ALIGATOR: EXPERIMENTS AND LIMITATIONS

4.1. Experiments

We tested Aligator on a number of interesting examples coming from academic and open source benchmarks. We summarize our results below, obtained on a machine with a 2.0GHz CPU and 2GB of RAM.

- Polynomial invariants have been inferred by Aligator in essentially no time, i.e. in an average less than 1 second, for programs implementing interesting algebraic algorithms, such

as for example GCD/LCM, square root and binary product computation, and Fibonacci algorithms. Altogether we used 25 loops for polynomial invariant generation. Aligator derived invariants for all examples we tried. When analysing our results, we note that the inferred polynomials are of degree up to 4, and the basis of the polynomial invariant ideal contains up to 5 polynomial invariant. Figure 2(a) shows the result of Aligator on a program implementing GCD/LCM computation.

• Quantified array invariants have been generated by Aligator on a large number of programs implementing array copies, shifts, initializations, and partitioning. All invariants were inferred within 1 second time limit. Altogether, we used about 680 loops over arrays and scalars. Aligator generated non-trivial scalar and quantified invariants for 332 loops. The loops that could not be handled by Aligator do not fulfil the P-solvable requirements, as they implement various sorting and permutation algorithms over arrays. When analysing our results, we note that the quantified invariants involve simple linear arithmetic properties over the array content, such as comparisons between array elements and constants. Figure 2(b) shows the result of Aligator on a program implementing an array (sign) partitioning algorithm.

```
x := a;y := b;u := b;v := a;
while (x≠y)
{
 if (x > y)
 then{x := x-y;v := v+u};
 else {y := y-x;u := u+v}
}
```

(a) P-solvable loop implementing the LCM/GCD computation of *a* and *b*.

The polynomial invariant inferred by Aligator is:

u*x+v*y-2*a*b=0

```
a:=b:=c:=0;
while (x<N,
{
if (A[a]≤0)
then {B[b]:= A[a];b:=b+1};
else {C[c]:=A[a];c:=c+1};
a:=a+1
}
```

(b) P-solvable loop, implementing an array partitioning algorithm.

The quantified array invariant inferred by Aligator is:

$$\bigwedge \begin{matrix}(\forall x)(0 \le x < b \Longrightarrow B[x] \ge 0)\\ (\forall x)(0 \le x < c \Longrightarrow C[x] < 0)\end{matrix}$$

Fig. 2: Aligator experiments.

4.2. Limitations

• **Loop tests** during scalar invariant generation are omitted. Polynomial invariants depending on loop guards are thus not inferred by Aligator. To this end, consider Figure 3(a). Aligator first rewrites this loop into Figure 3(b). Next, Aligator infers $xy + z - 1 = 0$ to be the polynomial loop invariant generating the polynomial invariant ideal of Figure 3(b), and hence of Figure 3(a) with omitted tests. Note however, that Figure 3(a) is semantically equivalent to the loop given in Example 1, as the path corresponding to the test condition $x > 10$ is never visited throughout the loop. According to Example 1, the invariant ideal of Figure 3(a) without omitting *tests* is generated by $xy - 1 = 0$. Hence the polynomial invariant ideals corresponding to Figure 3(a) with, respectively without loop tests are clearly different in power. As a further limitation, we also note that Aligator cannot infer disjunctive polynomial invariants. The completeness of Aligator in generating polynomial invariant ideals is thus crucially restricted to P-solvable loops where tests conditions are omitted.

```
x:=1; y:=1; z:=0;
while (x<5,
{
 if (x>10) then
   {z:=z+x;y:=y-1};
 x:=2*x; y:=y/2
}
```

(a) Loop

Aligator

1st step

```
x:=1; y:=1; z:=0;
while (...,
{
 if ... then
   {z:=z+x; y:=y-1};
 x:=2*x; y:=y/2
}
```

(b) Loop with omitted tests

Fig. 3. Limitations of Aligator on loops with omitted tests.

• **Scalar updates** of P-solvable loops are required to express linear recurrence equations with constant coefficients, where the order of the recurrence is at least 1. That is, scalar assignments are of the form $x_j := c * x + f(X)$ where X denotes the set of scalar loop variables, f is a polynomial expression over $X \backslash \{x_j\}$, and $c \neq 0$. Hence, loops with initializations, as given in Figure 4(a), are not P-solvable and cannot be handled by Aligator. A special class of P-solvable loops is the class of affine loops with no initializing assignments.

Another important expect of Aligator is that loop variables are abstracted to be rationals, so that algebraic techniques, such as Gröbner basis computation, can be performed.

```
x:=1; y:=1;
while (...,
{
 if ... then x:=0;
 x:=2*x; y:=y/2
}
```

(a) Loop with initialization. The assignment *x:=0* does not yield a recurrence of order greater than 1.

```
x:=1;
while (x<100,
{
  A[x+1]:= f(A[x]);
  x:=x+1
}
```

(b) Loop with read-and-write array A, where f is an uninterpreted function symbol.

Fig. 4. Limitations of Aligator on loop assignments.

• **Array updates** of P-solvable loops are made only to write-only arrays, where update expressions are over scalars and read-only array variables. Update expression might also contain some uninterpreted functions. P-solvable loops with read-and-write array variables, as in Figure 4(b), are thus not yet handled by Aligator. We also note that in case of P-solvable loops with nested conditionals, different write-only array variables and array indexes are used on different loop paths.

5. CONCLUSIONS

We describe experimental results and theoretical limitations on using Aligator for generating invariants of programs over scalars and arrays. Despite the theoretical limitations of Aligator, our experiments give practical evidence of the applicability of Aligator for loop reasoning and program verification. Future work includes addressing the current limitations of Aligator.

6. ACKNOWLEDGEMENTS

The first author is supported by a Hertha Firnberg Research grant (T425-N23) of FWF (Austrian Science Fund).

7. REFERENCES

Henzinger, T.; Hottelier, T.; Kovacs, L. & Rybalchenko, A. (2010) Aligators for Arrays. *Proc. of LPAR-17*, Indonesia, ISBN 978-3-642-16241-1, Fermüller, C. & Voronkov, A. (Eds.), pp. 348-356, Springer, Berlin

Kovacs, L. (2008) Reasoning Algebraically About P-Solvable Loops. *Proc. of TACAS*, Hungary, ISBN 978-3-540-78799-0, Ramakrishnan, C. R. & Rehof, J. (Eds.), pp. 249-264, Springer, Berlin

Gulwani, S.; Srivastava, S. & Venkatesan, R. (2009). Constraint-Based Invariant Inference over Predicate Abstraction. *Proc. of VMCAI*, USA, ISBN 978-3-540-93899-6, Jones, N. D. & Müller-Olm, M. (Eds.), pp. 120-135, Springer, Berlin

Rodriguez-Carbonell, E. & Kapur, D. (2007) Generating all Polynomial Invariants in Simple Loops. *J. Symbolic Computation*, Vol. 42, No. 4, pp. 443-476, ISSN 0747-7171