MLAT: Modal Logic Abstraction Tool

Toshifusa Sekizawa[1,2], Yoshinori Tanabe[1,3], Yoshifumi Yuasa[1], and Koichi Takahashi[1]

[1] National Institute of Advanced Industrial Science and Technology (AIST), Research Center for Verification and Semantics (CVS).
[2] Graduate School of Information Science and Technology, Osaka University.
[3] Graduate School of Information Science and Technology, The University of Tokyo.

MLAT: Modal Logic Abstraction Tool

Toshifusa Sekizawa[1,2], Yoshinori Tanabe[1,3], Yoshifumi Yuasa[1], and Koichi Takahashi[1]

[1] National Institute of Advanced Industrial Science and Technology (AIST), Research Center for Verification and Semantics (CVS)
[2] Graduate School of Information Science and Technology, Osaka University
[3] Graduate School of Information Science and Technology, The University of Tokyo

1 Overview

Verification tools based on abstraction technique have been developed, e.g., SLAM [1] and BLAST [2]. However, it is still difficult to analyze properties relating to heap manipulation. We present the tool MLAT (*Modal Logic Abstraction Tool*) for heap analysis based on predicate abstraction technique. Our approach [3] uses formulae of a modal logic to express predicates for abstraction. Related work of heap analysis is TVLA [4] and separation logic [5].

2 Aspect of a heap

We consider a heap as a set of *cells*, each of which contains pointers pointing to cells. To describe properties of heaps, we introduce a modal logic, which is a multi-modal extension of CTL, described below.

For each pointer-valued fields of a cell, we have a modality with forward and backward directions. Each boolean-valued field of a cell is used as a predicate symbol. Each pointer variable in a program is used as a nominal, or a special sort of predicate symbol that should be satisfied at only one state in a Kripke structure. We call the logic *2CTLN*, or two-way CTL with nominals. A heap can be viewed as a Kripke structure of the logic by regarding a cell as a state, and a "points-to" relation as a transition relation labelled with the corresponding field. The labelling function is also naturally defined: a predicate symbol is satisfied at a cell if its value is true and a nominal is satisfied if the corresponding variable points to it. Figure 1 shows a Kripke structure representing pointer variables point to cells.

In logics with nominals, we have formulae of the form $@_n\varphi$, where n is a nominal. That intuitively means that φ holds at the state where n satisfies at. In this paper, we call such formulae *pFormulae*.

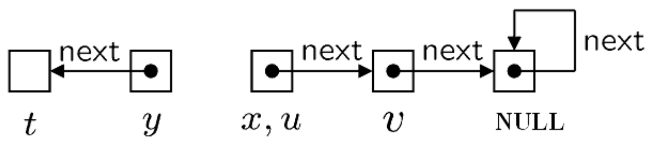

Figure 1: A Kripke structure representing pointer variables pointing to cells

3 Implementation

MLAT is a tool based on predicate abstraction [6] to generate a finite abstract transition system from a program that manipulates pointers. In MLAT, predicates for abstraction are described in pFormula. That makes various predicates expressible and gives concise expressions of complicated conditions. MLAT is implemented in Java, and uses JavaBDD [7] for better performance.

Input of MLAT is a file which consists of variable declarations, a program, predicates for abstraction, and properties for verification. Figure 2 is an input file by means of an example: *list reverse*. Output of MLAT is a file which includes the generated abstract transition system and properties for verification. The output file has a form compatible with NuSMV [8].

Programming language of MLAT is called PML (*Pointer Manipulation Language*), which is a simplified language but has enough expressive power of pointer manipulation. First, MLAT parses a PML program given in `Source` section in the input and generates a control flow graph (CFG) in which labels and commands are assigned to vertexes and edges respectively. Next, an abstract transition system is generated by tracing every path in the CFG. In predicate abstraction framework, two essential procedures are calculation of the weakest precondition and satisfiability checking. States in the abstract transition system are defined by combination of truth values of predicates and a vertex of CFG. The transition relation is decided by satisfiability checking. That is, if φ_1 and φ_2 are corresponding formulae for abstract states s_1 and s_2 respectively and σ is the command sequence from s_1 to s_2, transition from s_1 to s_2 exists if and only if $\varphi_1 \wedge \mathrm{wp}(\sigma, \varphi_2)$ is satisfiable, where $\mathrm{wp}(\sigma, \varphi_2)$ is the weakest precondition of φ_2 for σ. The authors proposed a procedure for the weakest preconditions for PML commands [3], and a decision procedure for 2CTLN [9]. After generation of the abstract transition system, MLAT invokes NuSMV and passes the output file to verify properties. Properties for verification are passed directly to the output file and not used in abstraction.

Automatic verification of large programs requires huge memory. We take an approach in cooperation with interactive proof assistant Agda [10]. That is, to prove a given property, divide it into lemmas for small subprograms on Agda. Such lemmas are then verified by MLAT that is connected by Agda plug-in system.

```
%%Decl  // variable declarations
  Var x, y, t, u, v;
  Field next;
  Label start, end;

%%Source  // a PML program
start:
  y := NULL;
  while (x != NULL) {
    t := y;
    y := x;
    x := x.next;
    y.next := t;
  }
end:

%%Pred  // predicates for abstraction
  p0 = x ==> E<next>F u;
  p1 = u ==> E<next>X v;
  p2 = v ==> NULL;
  p3 = x ==> E<next>F NULL;
  p4 = v ==> E<next>X u;
  p5 = v ==> x;
  p6 = y ==> u;

%%Spec  // specification to verify
  spec3 = [](start && p0 && p1 && !p2 && p3 -> [](end -> p4));
```

Figure 2: An input file of MLAT for verifying ‘list reverse’ program

4 An example

An example is the list reverse program. We verify that when the program takes a non-circular singly-linked list, it returns a list in which all pointers are reversed. Verified properties are as follows:

1. The program does not abort. That is, null pointers are not dereferenced.
   $\Box\,(\text{start} \wedge \neg\text{abort} \rightarrow \Box\neg\text{abort})$

2. All reachable cells from program variable x at the beginning are reachable from program variable y at the end.
   $\Box\,(\text{start} \wedge @_x\mathtt{E}_{\text{next}}\mathtt{F}u \wedge @_u\neg\text{NULL} \rightarrow \Box\,(\text{end} \rightarrow @_y\mathtt{E}_{\text{next}}\mathtt{F}u))$

3. For all cells u, v in the list, if the next cell of u is v at the beginning, then u is the next of v at the end.
   $\Box(\text{start} \wedge @_x\mathtt{E}_{\text{next}}\mathtt{F}u \wedge @_u\mathtt{E}_{\text{next}}\mathtt{X}v \wedge @_v\neg\text{NULL} \wedge @_x\mathtt{E}_{\text{next}}\text{FNULL}$
   $\rightarrow \Box\,(\text{end} \rightarrow @_v\mathtt{E}_{\text{next}}\mathtt{X}u))$

Property 1 and property 2 can be verified only with predicates which are subpFormulae of each property. If try to verify property 3 in the same way, spurious counterexamples are found. For verifying property 3 successfully, we need to add two more predicates. Figure 2 is the input file for verifying property 3 in which `p5` and `p6` are additional predicates. Verification times including NuSMV are; 1.2 seconds for property 1, 1.3 seconds for property 2, and 13.8 seconds for property 3. The measurements were performed using Red Hat Linux ES3 with Java 1.5.0_03 and NuSMV 2.4.1 on a computer equipped with Xeon 3.0GHz processor and 4GB RAM.

Property 3 is also proved interactively in cooperation with Agda. First, we manually divide property 3 into three lemmas. Each lemma is passed to MLAT and verified automatically. Finally, a proof of property 3 can be constructed by using the lemmas.

5 Conclusion

MLAT is ongoing work. We are working to deal with more practical programs. Future work for enhancement includes; abstract interpretation based on calculation of the strongest postcondition, a decision procedure for alternation free μ-calculus [11] for more expressive power of predicates, and refinement of predicates based on counterexamples.

Acknowledgements

This research was supported by Core Research for Evolutional Science and Technology (CREST) program "New High-Performance Information Processing Technology Supporting Information-Oriented Society" of Japan Science and Technology Agency (JST).

We are grateful to Makoto Takeyama for his design and implementation of the Agda plug-in for MLAT, and Toshinori Takai for his help in the early stage of MLAT.

References

[1] Thomas Ball and Sriram K. Rajamani. The SLAM project: debugging system software via static analysis. In *POPL*, pages 1–3, 2002.

[2] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Grégoire Sutre. Software verification with BLAST. In Thomas Ball and Sriram K. Rajamani, editors, *SPIN*, volume 2648 of *LNCS*, pages 235–239. Springer, 2003.

[3] Yoshinori Tanabe, Toshinori Takai, Toshifusa Sekizawa, and Koichi Takahashi. Preconditions of properties described in CTL for statements manipulating porinters. In *Supplemental Volume of the 2005 International Conference on DSN*, pages 228–234, 2005.

[4] Tal Lev-Ami and Shmuel Sagiv. TVLA: A system for implementing static analyses. In Jens Palsberg, editor, *SAS*, volume 1824 of *LNCS*, pages 280–301. Springer, 2000.

[5] John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *LICS*, pages 55–74. IEEE Computer Society, 2002.

[6] Susanne Graf and Hassen Saïdi. Construction of abstract state graphs with PVS. In Orna Grumberg, editor, *CAV*, volume 1254 of *LNCS*, pages 72–83. Springer, 1997.

[7] JavaBDD. Java library for manipulating BDDs. `http://javabdd.sourceforge.net/`.

[8] Alessandro Cimatti, Edmund M. Clarke, Fausto Giunchiglia, and Marco Roveri. NuSMV: A new symbolic model verifier. In Nicolas Halbwachs and Doron Peled, editors, *CAV*, volume 1633 of *LNCS*, pages 495–499. Springer, 1999.

[9] Yoshifumi Yuasa, Yoshinori Tanabe, Toshifusa Sekizawa, and Koichi Takahashi. A decision procedure for temporal formulas in automated predicate abstraction. In *The 22nd Japan Society for Software Science and Technology*, 2005. (in Japanese).

[10] Catarina Coquand. The AGDA proof system homepage, 1998. `http://www.cs.chalmers.se/~catarina/agda/`.

[11] Yoshinori Tanabe, Koichi Takahashi, Mitsuharu Yamamoto, Akihiko Tozawa, and Masami Hagiya. A decision procedure for the alternation-free two-way modal μ-calculus. In Bernhard Beckert, editor, *TABLEAUX*, volume 3702 of *LNCS*, pages 277–291. Springer, 2005.

MLAT: Modal Logic Abstraction Tool
(in English)
(算譜科学研究速報)
発行日：2007 年 2 月 19 日
編集・発行：独立行政法人産業技術総合研究所システム検証研究センター
同連絡先：〒563-8577 大阪府池田市緑丘 1-8-31
e-mail：informatics-inquiry@m.aist.go.jp
本掲載記事の無断転載を禁じます

MLAT: Modal Logic Abstraction Tool
(Programming Science Technical Report)
Feb. 19, 2007
Research Center for Verification and Semantics (CVS)
National Institute of Advanced Industrial Science and Technology (AIST)
1-8-31 Midorigaoka, Ikeda, Osaka, 563-8577, Japan
e-mail: informatics-inquiry@m.aist.go.jp
・Reproduction in whole or in part without written permission is prohibited.