Abstraction of programs in PML (Pointer Manipulation Language)

Koichi Takahashi
Yoshinori Tanabe
Toshifusa Sekizawa
Yoshifumi Yuasa
AIST
22 Sep 2005
Overview

• Research Interest: Abstraction of graph transformation systems using modal logics.
  – Garbage Collection, Cellular Automata
• Automatic verification tool for pointer manipulation programs
  – Main issue: abstraction of heap
• Use of modal logic to describe heap
  – Seeds for predicate abstraction are described in modal formula
• Development of abstraction tool based on this idea
Whole picture

Input

Program

Requirement Property

Output

OK

NG
Current Development

- Input:
  - Program
  - Requirement
  - Property
  - Abstraction Hints

- Abstract system generator
- Abstract system
- SPIN

- Output:
  - OK
  - NG

Abstraction Hints
Human Analysis
Counter Example

Diagram:
- Program → Abstract system generator → Abstract system → SPIN → OK
- Property → Abstraction Hints
- Abstraction Hints → Human Analysis
- Human Analysis → Counter Example
- Counter Example → NG

Abstract system generator and SPIN are highlighted in green.
Idea

• Predicate Abstraction Framework
  – Most of tools developed in the early days handle properties on the value of variables as predicates used in abstraction
  – It was difficult to express properties on the shape of the heap of programs

• We use modal formulas as a method for abstracting heap structures
  – another idea: separation logic?
Model of Heap: Pointer Structure

• Heap consists of cells
• Each cell has a pointer and a value
  – to simplify explanation
• Pointer variables
Pointer Structure as Kripke Structure

• Pointer Structure can be seen as a Kripke structure
• Atomic propositions are values and variables

AP = \{1,2,3,4,x,y,nil\}
2CTL

\[ \varphi ::= p \mid \neg \varphi \mid \varphi \lor \varphi \mid \varphi \land \varphi \mid E_A X \varphi \mid A_A X \varphi \mid E_A F \varphi \mid A_A F \varphi \mid E_A G \varphi \mid A_A G \varphi \]

where

\( p \): atomic proposition,
\( A \subseteq \text{Mod} \): set of modality,
\( \overline{a} \in \text{Mod} \),
\( \overline{a} = a \) for \( a \in \text{Mod} \).
Properties

• Many properties of heap can be described
• Confluence
  \[ x \land E_f F E_{\neg f} F y \]
• Reachable
  – \( x \) is reachable from \( y \)
  \[ y \rightarrow EF x \]
• Loop
  – \( x \) is in loop
  \[ x \rightarrow EXEF x \]
PML (pointer manipulation language)

• Target programs are written in PML
  – a tiny programming language manipulating heaps

• Statements are following:
  – \( x := y \)
  – \( x := y.next \)
  – \( x.next := y \)
  – \( x := \text{new}() \)
  – \( x\text{.val} := m \)
  – if (cond) goto line

• Dynamic logic for PML?
  – ongoing
a PML program example

0: y := nil
1: if (x == nil) goto 7
2: t := y
3: y := x
4: x := x.next
5: y.next := t
6: goto 1
7: (end)
a PML program example

0: y := nil
1: if (x == nil) goto 7
2: t := y
3: y := x
4: x := x.next
5: y.next := t
6: goto 1
7: (end)
a PML program example

0: y := nil
1: if (x == nil) goto 7
2: t := y
3: y := x
4: x := x.next
5: y.next := t
6: goto 1
7: (end)
a PML program example

0: y := nil
1: if (x == nil) goto 7
2: t := y
3: y := x
4: x := x.next
5: y.next := t
6: goto 1
7: (end)
a PML program example

0: y := nil
1: if (x == nil) goto 7
2: t := y
3: y := x
4: x := x.next
5: y.next := t
6: goto 1
7: (end)
a PML program example

0: $y := \text{nil}$
1: if ($x == \text{nil}$) goto 7
2: $t := y$
3: $y := x$
4: $x := x\text{.next}$
5: $y\text{.next} := t$
6: goto 1
7: (end)
a PML program example

0: y := nil
1: if (x == nil) goto 7
2: t := y
3: y := x
4: x := x.next
5: y.next := t
6: goto 1
7: (end)
a verification example

0: \( y := \text{nil} \)
1: \( \text{if } (x == \text{nil}) \text{ goto 7} \)
2: \( t := y \)
3: \( y := x \)
4: \( x := x.\text{next} \)
5: \( y.\text{next} := t \)
6: \( \text{goto 1} \)
7: \( \text{(end)} \)

- Verification statement:
  If a node is reachable from \( x \) at line 1, then the node is reachable from \( y \) at line 7.

\[
Q_1 = x \rightarrow EF \ u \quad Q_2 = y \rightarrow EF \ u
\]

\( Q_1 \) holds at line 1 \( \Rightarrow \ Q_2 \) holds at line 7.
a verification example

\[ Q_1 = x \rightarrow \text{EF} \ u \quad Q_2 = y \rightarrow \text{EF} \ u \]
a verification example

\[ Q_1 = x \rightarrow \text{EF } u \quad Q_2 = y \rightarrow \text{EF } u \]
a verification example

\[ Q_1 = x \rightarrow \text{EF } u \quad Q_2 = y \rightarrow \text{EF } u \]
Compute abstract transition

Program statement $s$

Concrete state

Abstract state $Q_1 \wedge Q_2$
Compute abstract transition

Concrete state

Abstract state

Weakest precondition

pre(s, Q1 \sqcup Q2)
Compute abstract transition

Abstract transition □

Intersection □

sat( ¬ Q1 □ Q2 □ pre(s,Q1 □ Q2)) = 1

Concrete state

program statement s

Abstract state
Q1 □ Q2

Weakest precondition
pre(s,Q1 □ Q2)
Compute abstract transition

No abstract transition

Disjoint

\[
sat(Q1 \sqcap Q2 \sqcap \text{pre}(s,Q1 \sqcap Q2)) = 0
\]

Concrete state

Abstract state

Weakest precondition

\[
\text{pre}(s,Q1 \sqcap Q2)
\]
Our case

• Weakest precondition of 2CTL for PML
• Sat checker for 2CTL

\[
Q_1 = \forall(x \rightarrow EF u) \\
\neg Q_2 = \forall(y \rightarrow \neg EF u)
\]

\[
\text{pre}(s, \neg Q_1) = \\
\forall(x \rightarrow EF EXx) \land \forall(u \rightarrow \neg(x \lor E (\neg x U (\neg x \land (EXx \lor y)))))) \\
\lor \forall(x \rightarrow \neg EF EXx) \land \forall(u \rightarrow \neg EF EXx)
\]

\[
\text{pre}(s, \neg Q_2) = \forall(u \rightarrow \neg(x \lor E (\neg x U (\neg x \land y)))) \\
(s = "t:=y; y:=x; x:=x.next; y.next:=t")
\]
Compute abstract transition

- Precondition and
- Satisfiability checking
Precondition

• We restrict 2CTL to p-formula for simplification
  – It is enough to describe important properties
• We have calculated weakest precondition of p-formula for each PML statement
  – weakest precondition of p-formula is p-formula
  – We are now implementing
Satisfiability check

• Usual: $\phi$ is sat $\phi$ there exists a Kripke structure $K$ s.t. $K \models \phi$
  – this checking is too rough
  – previous verification example does not work
Pointer Structure as Kripke Structure

- Pointer Structure can be seen as a Kripke structure
- Atomic propositions are values and variables
  \[ AP = \{1,2,3,4,x,y,nil\} \]

- Variable property holds at most one node
- A node has at most one next node
Satisfiability check

• Usual: $\phi$ is sat $\iff$ there exists a Kripke structure $K$ s.t. $K \models \phi$

• Our modification: $\phi$ is sat $\iff$ there exists a Pointer structure $P$ s.t. $P \models \phi$
  – more accurate
    • previous verification example works
  – We are now implementing
    • BDD