The Pointer Assertion Logic Engine

[PLDI '01]

Anders Møller
Michael I. Schwartzbach

Presented by K. Vikram
Cornell University

Introduction

- Pointer manipulation is hard
  - Find bugs, optimize code
- General Approach
  - Model the heap, including records & pointers
  - Describe properties in an assertion lang.
  - Verify correctness

The Objective

- Pointer Verification
- Check for
  - Type errors
  - Memory errors
  - Data structure invariant violations
- Safety critical data-type implementations

Some slides have been adapted from a talk by Anders Mϕller

Motivation

- Standard type-checking not enough
  - Tree $\equiv$ List
  - Avoiding memory errors
- Verify an abstract data type

Graph Types

- Regular Types
  - $T \rightarrow v(a_1 : T_1, \ldots, a_n : T_n)$
- Graph Types
  - $T \rightarrow v(\ldots\ a_i : T_i \ \ldots\ a_j : T_j\ [R]\ \ldots)$
  - data fields + routing fields (R)
  - backbone (ST) + other pointers
  - functionally dependent
- Doubly linked-list, trees with parent pointers, threaded trees, red-black trees, etc.

Routing Expressions

- Regular Expressions
- $\Downarrow a$ - move down along a
- $\uparrow$ - move along parent
- - verify presence at root
- \$ - verify presence at leaf
- T:v – verify type and variant
- Well-formedness

Examples

- List with first and last pointers

  H $\rightarrow$ (first: L, last: L[$\Downarrow$ first $\Downarrow$ tail * \$ $\uparrow$)

  L $\rightarrow$ (head: Int, tail: L)

  $\rightarrow$ ()

- Doubly-linked cyclic list

  D $\rightarrow$ (next: D, prev: D[$\uparrow$ + ^$\Downarrow$ next* \$]

  $\rightarrow$ (next: D[$\uparrow$ *], prev: D[$\uparrow$ + ^]

Monadic Second Order Logic

- Quantification done over predicates and terms
- Unary predicates
- Most expressive logic that is practically decidable
- Decidable using tree automata

Pointer Assertion Logic

- Monadic 2nd order
  - Over records, pointers and booleans
- Specify
  - Structural invariants
  - Pre- and post- conditions
  - Invariants and assertions

Methodology

- Verify a single ADT at a time
- Data structures as graph types
- Programs in a restricted language
- Annotations in Pointer Assertion Logic
  - properties of the store (assertions)
  - invariants
- Encode in monadic 2^{nd} order logic
- Use a standard tool (MONA)

Comparison with shape analysis

- Goals similar – approach different
- fixpoint iterations over store model vs. encoding of program in logic
- Similar precision and speed
- Use of loop invariants/assertions
- Need to specify operational semantics
- Restriction to graph types
- Generation of counter-examples

Routing Expressions

- Slightly generalized
- ptr <routingexp> ptr
- Up – x^T.p
- A general formula can be embedded

Example data structure

- Binary tree with pointers to root

```
type Tree = {
  data left,right:Tree;
  pointer root:Tree[root<(left+right)*>this &
                    empty(root^Tree.left union
                          root^Tree.right)];
}
```

Another Example

```
type Head = {
  data first: Node;
  pointer last:
    Node[this.first<next*.[pos.next=null]>last];
}
type Node = {
  data next: Node;
}
```

Example Program

```
proc concat(data l1,l2: Head): Head
{
  if (l1.last!=null) { l1.last.next = l2.first; }
  else { l1.first = l2.first; }

  if (l2.first!=null) { l1.last = l2.last; }
  l2.first = null;
  l2.last = null;
  return l1;
}
```

Details…

- Store Model
- Graph Types
- Abstract Programming Language
- Program Annotations

The Programming Language

typedecl	$\longrightarrow$	type T = { (*field* ;)* }
field	$\longrightarrow$	data $p^{\oplus}$: T
	$\mid$	pointer $p^{\oplus}$: T [*form*]
	$\mid$	bool $b^{\oplus}$
progvar	$\longrightarrow$	data $p^{\oplus}$: T
	$\mid$	pointer $p^{\oplus}$: T
	$\mid$	bool $b^{\oplus}$
procedure	$\longrightarrow$	proc n (*progvar*$^{\circledast}$) : (T $\mid$ void)
		(*logicvar* ;)*
		property
		({ (*progvar* ;)* *stm* })$^?$
		property

The Programming Language

$$
\begin{array}{lcl}
\mathit{stm} & \rightarrow & \mathit{stm}\ \mathit{stm} \\
 & | & \mathit{asn}^{\oplus}\ ; \\
 & | & \mathit{proccall}\ ; \\
 & | & \texttt{if}\ (\ \mathit{condexp}\ )\ \{\ \mathit{stm}\ \}\ (\ \texttt{else}\ \{\ \mathit{stm}\ \}\ )^{?} \\
 & | & \texttt{while}\ \mathit{property}\ (\ \mathit{condexp}\ )\ \{\ \mathit{stm}\ \} \\
 & | & \texttt{return}\ \mathit{progexp}\ ; \\
 & | & \texttt{assert}\ \mathit{property}\ ; \\
 & | & \texttt{split}\ \mathit{property}\ \mathit{property}\ ; \\
\mathit{asn} & \rightarrow & \mathit{lbexp} = (\ \mathit{condexp}\ \mid\ \mathit{proccall}\ ) \\
 & | & \mathit{lptrexp} = (\ \mathit{ptrexp}\ \mid\ \mathit{proccall}\ )
\end{array}
$$

The Programming Language

```
condexp  →  bexp  |  ?  |  [ form ]
bexp     →  ( bexp )              |  ! bexp
         |  bexp & bexp           |  bexp | bexp
         |  bexp => bexp          |  bexp <=> bexp
         |  bexp = bexp           |  ptrexp = ptrexp
         |  bexp != bexp          |  ptrexp != ptrexp
         |  true  |  false        |  lbexp
lbexp    →  b  |  ptrexp . b
ptrexp   →  null  |  lptrexp
lptrexp  →  p  |  ptrexp . p
proccall →  n ( ( condexp | ptrexp )^⊛ ) [ formula ]
```

Program Annotations

- Monadic 2^{nd} order Logic on graph types
- Quantification over heap records
  - Individual elements, sets of elements
- Formulas* used for
  - Constraining destinations of pointers
  - Invariants in loops and procedure calls
  - Pre- and post- conditions
  - Assert and split statements

Monadic 2L on finite trees

- $\Phi ::= \neg\, \Phi \mid \Phi \vee \Phi \mid \Phi \wedge \Phi \mid \Phi \Rightarrow \Phi \mid$
  $\Phi \Leftrightarrow \Phi \mid \forall^1 x.\Phi \mid \exists^1 x.\Phi \mid \forall^2 x.\Phi \mid \exists^2 x.\Phi \mid$
  $t = t \mid t \in T \mid T = T \mid T \subseteq T \mid \ldots$
  (formulas)
- $T ::= X \mid T \cup T \mid T \cap T \mid T \setminus T \mid \varnothing$
  (set terms)
- $t ::= x \mid t.\mathrm{left} \mid t.\mathrm{right} \mid t.\mathrm{up}$
  (position terms)

Program Annotations

form → (existpos | allpos) $p^{\oplus}$ of T : *form*

| (existset | allset) $s^{\oplus}$ of T : *form*

| (existptr | allptr) $p^{\oplus}$ of T : *form*

| (existbool | allbool) $s^{\oplus}$: *form*

| (*form*) | ! *form*

| *form* & *form* | *form* | *form*

| *form* => *form* | *form* <=> *form*

| *ptrexp* in *setexp* | *setexp* sub *setexp*

| *setexp* = *setexp* | *setexp* != *setexp*

| empty (*setexp*) | *bexp*

| return | *n* . *b*

| *m* ((*form* | *ptrexp* | *setexp* $)^{\circledast}$)

| *ptrexp* < *routingexp* > *ptrexp*

predicate → pred *m* (*logicvar*$^{\circledast}$) = *form*

Program Annotations

$$
\begin{array}{lcl}
\textit{logicvar} & \rightarrow & \texttt{pointer}\ p^{\oplus}\ :\ T \\
 & \mid & \texttt{bool}\ b^{\oplus} \\
 & \mid & \texttt{set}\ s^{\oplus}\ :\ T
\end{array}
$$

$$
\textit{ptrexp} \rightarrow \ \ldots \ \mid \ \texttt{return} \ \mid \ n\ .\ p
$$

$$
\begin{array}{lcl}
\textit{setexp} & \rightarrow & s \\
 & \mid & \textit{ptrexp}\ \hat{}\ T\ .\ p \\
 & \mid & \{\ \textit{ptrexp}^{\oplus}\ \} \\
 & \mid & \textit{setexp}\ \texttt{union}\ \textit{setexp} \\
 & \mid & \textit{setexp}\ \texttt{inter}\ \textit{setexp} \\
 & \mid & \textit{setexp}\ \texttt{minus}\ \textit{setexp}
\end{array}
$$

A More Involved Example

- Threaded Trees
  - Pointer to successor in post-order traversal

```
type Node = {
  data left,right:Node;
  pointer post:Node[POST(this,post)];
  pointer parent:Node[PARENT(this,parent)];
}
```

```
a^Node.left union a^Node.right={b}
```

The Fix Procedure

```
proc fix(pointer x: Node): void
{
  if (x.left=null & x.right=null) {
    if (x.parent=null) { x.post = x; }
    else {
      if (x.parent.right=null | x.parent.right=x) {
        x.post = x.parent;
      }
      else {
        x.post = findsmallest(x.parent.right);
      }
    }
  }
  else { x.post = findsmallest(x); }
}
```

Annotations

Precondition to fix

```
[x!=null {Node.post[ALMOSTPOST(this,post,x)]}]
```

ALMOSTPOST Predicate

```
(this!=x => POST(this,post)) & (this=x => post=null)
```

Hoare Triples

- {P} S {Q}
- P = pre-conditions (boolean predicate)
- Q = post-conditions (-do-)
- S = program
- Standard tools available

Verification with Hoare Logic

- Split program into Hoare triples
  "property stm"
- Use PAL as assertion language
- Cut-points
  - beginning/end of procedure and while bodies
  - split statements
  - graph types valid only at cut-points

Verification with Hoare Logic

- Hoare Triple – property stm
- stm is without loops, procedure calls
- Use *transduction* to simulate statements
- Update store predicates (11 kinds)
- Interface for querying the store

Advantages over earlier tools

- Can handle temporary violations
  - Overriding pointer directives
  - Allow different constraints at different points
  - Use property instead of formula
- Modular and thus, scalable

MONA

- Reduces formulas to tree automata
- Deduces validity or generates counter-examples

Evaluation

- As fast as previous tools
- Very few intractable examples in practice
- Found a null-pointer dereference in a bubblesort example

Evaluation

Example name	Lines of code	Invariants (formulas)	GTA operations	Largest GTA		Time (seconds)	Memory (MB)
				States	BDD nodes		
reverse	16	1	1,109	35	142	0.52	2
search	12	1	853	27	85	0.25	2
zip	33	1	1,753	174	730	4.58	11
delete	22	0	973	73	349	1.36	5
insert	33	0	1,005	103	443	2.66	7
rotate	11	0	590	44	213	0.22	1
concat	24	0	1,056	48	177	0.47	3
bubblesort_simple	43	1	1,477	373	3,289	2.86	18
bubblesort_boolean	43	2	1,737	357	3,922	3.37	12
bubblesort_full	43	2	2,069	373	3,291	4.13	19
orderedreverse	24	1	1,091	29	100	0.46	3
recreverse	15	2	1,019	42	176	0.34	2
doublylinked	72	1	4,163	230	796	9.43	13
leftrotate	30	0	1,489	165	1,550	4.62	7
rightrotate	30	0	1,489	165	1,550	4.68	7
treeinsert	36	1	1,989	137	844	8.27	31
redblackinsert	57	7	4,279	297	2,419	35.04	44
threaded	54	4	3,505	50	248	3.38	7

Finally

- Questions
- Comments
- Praise/Criticism
- Thank you!