An Abstract Machine for the Old Value Retrieval

Piotr Kosiuczenko

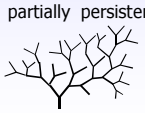
ISI, WAT
Warsaw

Eiffel Approach to `@pre/old`

- Restrict post-conditions to formulas of the form:
  $t[(t_1)@pre/x_1, \ldots, (t_n)@pre/x_n]$
- Compute and save values of $t_1,\ldots, t_n$ when the constrained method is called
- All implementations of `@pre` follow this approach: e.g. JML and USE, DOT, …

Problems with this approach

- Syntax restrictions
- A potential increase of computational complexity
  `if 1+1=2 then true else` **`q@pre`** `endif`
- Extensive cloning of system states - collections are the major slowdown factor
- The lack of transparency in respect to object identity
- Alternatively one could try to use the so the called fat structures as they are used in case of partially persistent structures



Solution Idea

- Number method calls – a larger number means a later call
- Define a history stack for every relevant attribute a to store a's snapshots
- Save a snapshot of a in its history stack, when a is modified for the first time during a call of a method

The Formal Model

- We model states of an oo-system and the corresponding transitions by an abstract machine
- States comprise the call-stack and the corresponding heap, and also:
  - method call number
  - heap history
  - attributes' history function

Model of OO Computation: States

- *Set of attributes* and *object locations*
  $A = \{a_1,\ldots, a_n\}$, OL
- *Object store (store) - heap state*
  $OS =_{df} \{os : A \times OL \not\to OL_?\}$
- *Store history*
  $SH =_{df} (OS \times OP \times N)^+$
- *Attribute snpshot*
  $H =_{df} (OL \times N)^*$
- *Attribute history*
  $AH =_{df} \{h : A \times OL \not\to H_?\}$
- *Computation state*
  $CS =_{df} SH \times AH \times N$
- *Initial state*
  $inSt =_{df} ((st_?, main, 0), h_?, 0)$

Model of OO Computation: Transitions

Let cs = (sh, h, n), cs' = (sh', h', n'),
sh = $sh_0(st_k, op_k, n_k)$, $sh_0 = (st_0, main, 0)...(st_{k-1}, op_{k-1}, n_{k-1})(st_k, op_k, n_k)$
h(a, o) = ? Æ h(a, o) ≠ ²) h(a, o) = $ah_0(o_l, r_l)$,

$ah_0 = (o_0, r_0) ... (o_{l-1}, r_{l-1})$

- cs $R_{call(op)}$ cs' , n' = n+1 Æ sh' = sh(st, op, n+1) Æ h' = h
- cs $R_{set(a, o, o')}$ cs' , n' = n Æ sh' = sh_0(st ', op, m) Æ st ' = st[(a, o) ↦ o'] Æ
  - h(a, o) = ?) h' = h[(a, o) ↦ (o', n_k)]
  - h(a, o) = ² Ç r_l < n_k Æ st_k(a, o) ≠ o_l)

    h' = h[(a, o) ↦ h(a, o)(st_k(a, o), n_k)]
  - r_l < n_k Æ st_k(a, o) = o_l)

    h' = h[(a, o) ↦ ah_0(o_l, n_k)]
  - h' = h in other cases

Model of OO Computation: Transitions cont.

- cs $R_{return(op)}$ cs' , sh' = (st_0, main, 0) ... (st_k, op_{k-1}, n_{k-1})
  - |sh_0| > 0) h' = h Æ n' = n
  - |sh_0| = 0) $8_{a\ 2\ A,\ o\ 2\ OL}$ h'(a, o) = if h(a, o) ≠ ? then ² else ? Endif Æ n' = 0
- cs $R_{cleanTop(a, o)}$ cs' , 1 < |h(a, o)| Æ n_k 6 r_{l-1} Æ sh' = sh Æ n' = n Æ h' = h[(a, o) ↦ ah_0]
- cs $R_{cleanWhole(a, o)}$ cs' , 1 < |h(a, o)| Æ sh' = sh Æ n' = n Æ h' = h[(a, o) ↦ ah], where ah is obtained from ah_0 by removing snapshots with time-stamps not being a smallest upper bound of an unterminated method call number

Invariant

For cs = (sh, h, n) let sh = (st_0, main, 0)(st_1, op_1, n_1) ... (st_k, op_k, n_k).
For h(a, o) ≠ ?, we assume that h(a, o) = (o_1, r_1) ... (o_l, r_l).

- $8_{0\ 6\ i\ <\ k}$ st_i(a, o) ≠ ?) st_{i+1}(a, o) ≠ ?
- st_k(a, o) = ? , h(a, o) = ?
- h(a, o) ≠ ?) 0 < n_1 < ... < n_k 6 n
- h(a, o) ≠ ?) 0 < r_1 < ... < r_l 6 n
- $8_{0\ 6\ i\ <\ k}$? ≠ st_i(a, o) ≠ st_{i+1}(a, o)) $9_{1\ 6\ j\ 6\ l}$ n_{i+1} 6 r_j
- $8_{0\ 6\ i\ <\ k,\ 0\ 6\ j\ 6\ l}$ st_i(a, o) ≠ ? Æ n_{i+1} 6 r_j Æ (1 6 j-1) r_{j-1} < n_{i+1}))

  st_i(a, o) = o_j

Invariant Preservation

Theorem

Inv is satisfied in the initial state inSt.
If cs ² Inv and cs R_{step} cs', where step is one of the transition steps, then cs' ² Inv.

Lemma

Let cs be a computation state as described in the invariant.
If cs ² Inv and st_i(a, o) ≠ ?, then
st_{i+1}(a@pre, o) =

if n_{i+1} 6 r_j Æ (0 < j - 1) r_{j-1} < n_{i+1}) then o_j else st_{i+1} (a, o) endif

Flattening Method Calls

csa is a *non-flattened computation state* and csb is a *flattened state* if, and only if, the following conditions are satisfied:

- csa = ((stb_0, main, 0)ih0 ... (sta_m, rop_m, na_m)ih_m, ha, na)
- csb = ((stb_0, main, 0)(stb_1, rop_1, nb_1) ... (stb_m, rop_m, nb_m), hb, nb)

A non-flattened state csa is *equivalent* to a flattened state csb (csa ¼ csb) if, and only if, the following conditions are satisfied:

- If |ih_j| > 0, then the last store in ih_j equals stb_j, for j = 0,..., m
- If ih_j = 0, then sta_j = stb_j, for j = 0,..., m

Weak Bisimulation

Theorem

¼ is a kind of weak bisimulation relation; i.e. inSt ¼ inSt, and
if csa ¼ csb, then the following conditions hold:

- iop 2 IOp Æ csa ⟶$_{call(iop)}$ csa') csa' ¼ csb
- rop 2 ROp Æ csa ⟶$_{call(rop)}$ csa')

  $9_{csb'}$ csb ⟶$_{call(rop)}$ csb' Æ csa' ¼ csb'
- . . .
- rop 2 ROp Æ csb ⟶$_{return(rop)}$ csb')

  $9_{csa_0,..., csa_n, csa'}$ csa = csa_0 Æ
  - csa_i ⟶$_{return(iop_i)}$ csa_{i+1} Æ csa_{i+1} ¼ csb, for i = 0,..., n-1, Æ
  - csa_n ⟶$_{return(rop)}$ csa' Æ csa' ¼ csb'

Weak Bisimulation

- csa ⟶$_{clean*(a, o)}$ csa') csa' ¼ csb
- . . .
- iop 2 IOp Æ csa ⟶$_{return(iop)}$ csa') csa' ¼ csb
- rop 2 ROp Æ csa ⟶$_{return(rop)}$ csa')
  9$_{csb'}$ csb ⟶$_{return(rop)}$ csb' Æ csa' ¼ csb'

Weak Bisimulation

Lemma

Let csa, csb be computation states of the above form such that csa ¼ csb. For j = 1,..., m, if op$_j$ is a relevant method, then sta$_j$(a@pre, o) = stb$_j$(a@pre, o).

Sufficiently Persistent Structures

- We call a *structure suffciently* persistent, if for every non-terminated method call its version from before the call can be accessed, but only the most recent version can be updated
- Sufficiently persistent structures are related to the so called semi persistent structures as partially persistent structures to persistent ones
- The defined abstract machine corresponds to a general form of sufficiently persistent structures

Sufficiently Persistent Structures

- The old value archiving does not increase the complexity of the instrumented methods
- The average time of old value retrieval is constant
- It can be ensured without an increase method's time-complexity that the history size is for every archived attribute is of the linear order in respect to the call-stack size

Example: Hanoi Towers

- The standard solution relies on a recursive algorithm and requires an exponential number of moves in respect to n
- The corresponding partially persistent structure has an exponential size order with respect to n
- The call-stack size is maximally n; consequently, we can ensure that the sufficiently persistent structure has at most the size order n^2

Concluding Remarks

- We defined an abstract machine for computing `@pre`
- There is an invariant preserved by this machine, which implies that the `@pre` value is computable on the basis of history attributes
- Calls of irrelevant methods can be abstracted away
- We defined the notion of sufficiently persistent structure and showed that they are more adequate in this case than partially persistent ones