Completeness of Pointer Program Verification by Separation Logic

Makoto Tatsuta (National Institute of Informatics)
joint work with:
Wei-Ngan Chin (National Univ of Singapore)
Mahmudul Faisal Al Ameen (Graduate Univ for Advanced Studies)

Seminar

School of Computer Science and Engineering, Seoul National University
March 15, 2010

Introduction

Pointer programs
- while programs + heaps
- memory allocation, lookup, mutation, and dispose

Separation logic
- Peano arithmetic + heap primitives + separating connectives

Completeness: Every correct program is proved to be correct

Question: Is separation logic for pointer programs complete?

Results:
(1) Completeness of separation logic for pointer programs
(2) Expressiveness of separation logic for pointer programs
(3) Completeness of that under deterministic semantics

Ideas:
- Relative completeness and expressiveness for while programs
- A formula that exactly describes the current heap

Background 1/7: While Programs

While Programs $P ::= x := e | \text{if } (b) \text{ then } (P) \text{ else } (P) |$
$\quad \text{while } (b) \text{ do } (P) | P; P$

Eg. while $(x < 11)$ do $(y := y + x; x := x + 1)$

Semantics $[\![P]\!]$

- a store s : variables → natural numbers
- $[\![P]\!](s_1) = s_2$

Eg.
$s_1(x) = 1,\ s_1(y) = 0,\ s_2(x) = 11, s_2(y) = 55$
$[\![\text{while } (x < 11) \text{ do } (y := y + x; x := x + 1)]\!](s_1) = s_2$

Background 2/7: Asserted Programs

Assertions A formulas in Peano arithmetic

Eg. $x = 11 \wedge y = 55$

Asserted Programs $\{A\}P\{B\}$

$\{A\}P\{B\}$ is true iff
- If the initial state that satisfies A and the execution of the program P terminates, then the resulting state satisfies B
- Partial correctness

Eg.
$\{x = 1 \wedge y = 0\}$while $(x < 11)$ do $(y := y + x; x := x + 1)\{x = 11 \wedge y = 55\}$
is true

Background 3/7: Hoare Logic

Boolean expression b a quantifier-free assertion

Inference rules:

$$\frac{}{\{A[x := e]\}x := e\{A\}}\,(assignment)$$

$$\frac{\{A \wedge b\}P_1\{B\} \quad \{A \wedge \neg b\}P_2\{B\}}{\{A\}\text{if }(b)\text{ then }(P_1)\text{ else }(P_2)\{B\}}\,(if)$$

$$\frac{\{A \wedge b\}P\{A\}}{\{A\}\text{while }(b)\text{ do }(P)\{A \wedge \neg b\}}\,(while)$$

$$\frac{\{A\}P_1\{C\} \quad \{C\}P_2\{B\}}{\{A\}P_1; P_2\{B\}}\,(comp)$$

$$\frac{\{A_1\}P\{B_1\}}{\{A\}P\{B\}}\,(conseq) \quad (A \to A_1,\ B_1 \to B\ \text{true})$$

$\{A\}P\{B\}$ is provable

- There exists its derivation by the inference rules

Example

$\{x = 1 \wedge y = 0\}$while $(x < 11)$ do $(y := y + x; x := x + 1)\{x = 11 \wedge y = 55\}$
is provable

Let the loop invariant I be $x \leq 11 \wedge y = 1 + \ldots + (x - 1)$
$x = 1 \wedge y = 0 \rightarrow I$ is true
$I \wedge x \geq 11 \rightarrow x = 11 \wedge y = 55$ is true

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \{I \wedge x < 11\}y := y + x; x := x + 1\{I\}\end{array}}{\{I\}\text{while } (x < 11) \text{ do } (y := y + x; x := x + 1)\{I \wedge x \geq 11\}}\ (while)}{\{x = 1 \wedge y = 0\}\text{while } (x < 11) \text{ do } (y := y + x; x := x + 1)\{x = 11 \wedge y = 55\}}\ (conseq)$$

Background 4/7: Soundness and Completeness

Soundness: if $\{A\}P\{B\}$ is provable, $\{A\}P\{B\}$ is true
- If the system proves a program is correct, it is indeed correct
- Significance of a verification system

Completeness: if $\{A\}P\{B\}$ is true, $\{A\}P\{B\}$ is provable
- If a program is correct, the system surely proves it is correct
- The converse of Soundness
- Ability of a verification system

Hoare Logic is sound and complete

Background 5/7: Pointer Programs

Pointer programs   while programs with heaps

Programs $P ::= x := e|$if (b) then (P) else $(P)|$
while (b) do $(P)|P; P|$
$x :=$ cons$(e, e)|x := [e]|[e] := e|$dispose$(e)$

Eg.

Allocation $x :=$ cons$(0, 3)$ x:$\boxed{100}$ 100:$\boxed{0}$ 101:$\boxed{3}$
Lookup $x := [101]$ x:$\boxed{3}$ 100:$\boxed{0}$ 101:$\boxed{3}$
Mutation $[100] := 2$ x:$\boxed{3}$ 100:$\boxed{2}$ 101:$\boxed{3}$
Dispose dispose(100) x:$\boxed{3}$ 101:$\boxed{3}$

Background 6/7: Semantics

N the set of natural numbers (values and addresses)
Locs $= \{n \in N | n > 0\}$
0 a null pointer

a store s:variables $\rightarrow N$
a heap h:Locs $\rightarrow_{fin} N$
a state (s, h)

A program (1) terminates without abort (normal execution), (2) terminates with abort (memory error), or (3) does not terminate
- abort: referring to an unallocated address e in $x := [e]$, $[e] := e_2$, or dispose(e)

Semantics $[\![P]\!]((s, h))$ a set of states
- If the initial state is (s, h) and the execution of the program P terminates without abort, then $P((s, h))$ is the set of possible resulting states
- nondeterministic: a choice of free memory for $x :=$ cons(e_1, e_2)

Background: Semantics (cont.)

$[\![P]\!](\text{abort}) = \{\text{abort}\},$

$[\![x := e]\!]((s,h)) = \{(s[x := [\![e]\!]_s], h)\},$

$[\![\text{if } (b) \text{ then } (P_1) \text{ else } (P_2)]\!]((s,h)) =$

$\quad [\![P_1]\!]((s,h)) \text{ if } [\![b]\!]_s = \text{true},$

$\quad [\![P_2]\!]((s,h)) \text{ otherwise},$

$[\![\text{while } (b) \text{ do } (P)]\!]((s,h)) = \{(s,h)\} \text{ if } [\![b]\!]_s = \text{false},$

$\quad [\![\text{while } (b) \text{ do } (P)]\!]([\![P]\!]((s,h))) \text{ otherwise},$

$[\![P_1; P_2]\!]((s,h)) = [\![P_2]\!]([\![P_1]\!]((s,h))),$

$[\![x := \text{cons}(e_1, e_2)]\!]((s,h)) =$

$\quad \{(s[x := n], h[n := [\![e_1]\!]_s, n+1 := [\![e_2]\!]_s])|$

$\quad n > 0, n, n+1 \notin \text{Dom}(h)\},$

$[\![x := [e]]\!]((s,h)) =$

$\quad \{(s[x := h([\![e]\!]_s)], h)\} \text{ if } [\![e]\!]_s \in \text{Dom}(h),$

$\quad \{\text{abort}\} \text{ otherwise},$

$[\![[e_1] := e_2]\!]((s,h)) =$

$\quad \{(s, h[[\![e_1]\!]_s := [\![e_2]\!]_s])\} \text{ if } [\![e_1]\!]_s \in \text{Dom}(h),$

$\quad \{\text{abort}\} \text{ otherwise},$

$[\![\text{dispose}(e)]\!]((s,h)) =$

$\quad \{(s, h|_{\text{Dom}(h) - \{[\![e]\!]_s\}})\} \text{ if } [\![e]\!]_s \in \text{Dom}(h),$

$\quad \{\text{abort}\} \text{ otherwise.}$

Background 7/7: Separation Logic

Reynolds (LICS 02)

Assertions for pointer programs

Assertions $A ::= \mathsf{emp} | e = e | e < e | e \mapsto e | \neg A | A \wedge A | A \vee A | A \to A | \forall x A | \exists x A | A * A | A \mathrel{-\!\!*} A$

Eg.

emp the current heap is empty

$3 \mapsto 5$ the current heap is 3:$\boxed{5}$ (the current heap is a single cell)

$A * B$ the current heap can be split into some two heaps h_1 and h_2 such that A is true for h_1 and B is true for h_2

$A \mathrel{-\!\!*} B$ if the heap h satisfies A, then B is true for the heap obtained from the current heap by adding h

Background: Separation Logic (cont.)

Inference rules:

$$\frac{}{\{\forall x'((x' \mapsto e_1, e_2) \mathrel{-\!\!*} A[x := x'])\} x := \mathsf{cons}(e_1, e_2)\{A\}}\ (cons)$$

$$(x' \notin \mathsf{FV}(e_1, e_2, A))$$

$$\frac{}{\{\exists x'(e \mapsto x' * (e \mapsto x' \mathrel{-\!\!*} A[x := x']))\} x := [e]\{A\}}\ (lookup)$$

$$(x' \notin \mathsf{FV}(e, A))$$

$$\frac{}{\{(\exists x(e_1 \mapsto x)) * (e_1 \mapsto e_2 \mathrel{-\!\!*} A)\}[e_1] := e_2\{A\}}\ (mutation)$$

$$(x \notin \mathsf{FV}(e_1))$$

$$\frac{}{\{(\exists x(e \mapsto x)) * A\}\mathsf{dispose}(e)\{A\}}\ (dispose)$$

$$(x \notin \mathsf{FV}(e))$$

Related Work

[Reynolds 02(LICS)]
- gave separation logic and showed its soundness

[Nguyen, David, Qin, and Chin 07]
- Verification system based on separation logic
- Implemented
- Automatic verification of the quick sort program in a second

[Berdine, Calcagno, and O'Hearn 05]
- Verification system based on separation logic
- Implemented

[Ishtiaq and O'Hearn 01(POPL)]
- Completeness only for programs without if-statements nor while-statements

No completeness results

Our result: a proof of the completeness (SEFM09)

Main Theorem

Theorem (Completeness). If $\{A\}P\{B\}$ is true, then $\{A\}P\{B\}$ is provable

Ideas:
- Extending the completeness proof of Hoare Logic
- Relative completeness: We assume all true assertions

$$\frac{\{A_1\}P\{B_1\}}{\{A\}P\{B\}}(conseq) \quad (A \to A_1,\ B_1 \to B \text{ true})$$

- Weakest precondition: For a given program P and a given assertion B, the weakest precondition of P and B is the weakest condition for the input states such that B is true after the execution of P

Difficulty:
- Expressiveness: For a given program P and a given assertion B, there exists some assertion that describes the weakest precondition of P and B

Assertion for Current Heap

(n, m) the code that represents the pair of natural numbers n, m

$\langle n_1, \ldots, n_k \rangle$ the code that represents the sequence $n_1, \ldots, n_k$

Lookup(x, y, z) The sequence x contains the pair (y, z)

- Lookup(x, l, k) iff $x = \langle n_1, \ldots, (l, k), \ldots, n_m \rangle$

$$\text{Heap}(m) = \forall xy(\text{Lookup}(m, x, y) \leftrightarrow (x \mapsto y * 0 = 0))$$

Heap$(\langle (l_1, n_1), \ldots, (l_k, n_k) \rangle)$ means:

Dom$(h) = \{l_1, \ldots, l_k\}$ and $h(l_i) = n_i$ where h is the current heap

(The current heap is $l_1 {:} \boxed{n_1}$ . . . $l_k {:} \boxed{n_k}$)

Expressiveness Theorem

Store$_{x_1,\ldots,x_n}(\langle m_1,\ldots,m_n\rangle)$ means $s(x_i) = m_i$ where s is the current store
The code $\lceil(s,h)\rceil$ of (s,h) is $(\lceil s\rceil, \lceil h\rceil)$
Vector notation $\overrightarrow{x} = x_1,\ldots,x_n$

Exec$_{P,\overrightarrow{x}}(\lceil r_1\rceil, \lceil r_2\rceil)$ means $[\![P]\!](r_1) \ni r_2$
($\overrightarrow{x}$ includes free variables in P)

$$\mathsf{W}_{P,A}(\overrightarrow{x}) = \forall xyzw(\mathsf{Store}_{\overrightarrow{x}}(x) \wedge \mathsf{Heap}(y) \wedge \mathsf{Pair2}(z,x,y) \wedge \mathsf{Exec}_{P,\overrightarrow{x}}(z,w) \to w > 0 \wedge \exists y_1 z_1(\mathsf{Pair2}(w,y_1,z_1) \wedge \mathsf{Eval}_{A,\overrightarrow{x}}(y_1,z_1)))$$

($\overrightarrow{x}$ includes free variables in P and A)

Theorem (Expressiveness). W$_{P,A}(\overrightarrow{x})$ describes the weakest precondition of P and A

Proof of Completeness Theorem

By induction on P.
Cases according to the last rule.

Case $(comp)$.

$$\frac{\{A\}P_1\{C\} \quad \{C\}P_2\{B\}}{\{A\}P_1; P_2\{B\}}\,(comp)$$

Suppose $\{A\}P_1; P_2\{B\}$ is true.
Let C be $\mathsf{W}_{P_2,B}(\overrightarrow{x})$ where $\overrightarrow{x}$ includes free variables in B, P_2.
By Expressiveness Theorem, $\{A\}P_1\{C\}$ and $\{C\}P_2\{B\}$ are both true.
By induction hypothesis for P_1 and P_2, $\{A\}P_1\{C\}$ and $\{C\}P_2\{B\}$ are both provable.
By $(comp)$, $\{A\}P_1; P_2\{B\}$ is provable.

Expressiveness Theorem is also necessary for the case of $\{A\}$while (b) do $(P)\{B\}$. $\square$

Deterministic Semantics 1

In the previous semantics, $x :=$ cons(e_1, e_2) finds some new memory cells, which we do not know. This is formalized by nondeterminism.

Deterministic semantics specifies the new memory cells. For simplicity, we assume the free memory cells will be chosen so that the address is smallest among free memory cells.

Assertions $A ::= ...|$ New(e)

New(e) means to hold if and only if e is the address of the first free cells in memory space.

Semantics $[\![P]\!](r_1) = r_2$
- the execution of P with the initial state r_1 terminates with the resulting state r_2

$$[\![x := \text{cons}(e_1, e_2)]\!]((s, h)) = (s[x := n], h[n := [\![e_1]\!]_s, n + 1 := [\![e_2]\!]_s]),$$

where n is the smallest number such that
$n > 0$ and $n, n + 1 \notin \text{Dom}(h)$

Deterministic Semantics 2

Our new inference rule:

$$\frac{}{\{\exists x'(\mathsf{New}(x') \wedge ((x' \mapsto e_1, e_2) -\!\!* A[x := x']))\} x := \mathsf{cons}(e_1, e_2)\{A\}} \; (cons)$$

$$(x' \notin \mathsf{FV}(e_1, e_2, A))$$

Note. The rule $(conseq)$ assumes all true assertions for New

Theorem(Soundness). If $\{A\}P\{B\}$ is provable in the system with New(e), then $\{A\}P\{B\}$ is true under deterministic semantics

Theorem(Completeness). If $\{A\}P\{B\}$ is true under deterministic semantics, then $\{A\}P\{B\}$ is provable in the system with New(e)

Conclusion

Pointer programs
- while programs + heaps
- memory allocation, lookup, mutation, and dispose

Separation logic
- Peano arithmetic + heap primitives + separating connectives

Completeness: Every correct program is proved to be correct

Question: Is separation logic for pointer programs complete?

Results:
(1) Completeness of separation logic for pointer programs
(2) Expressiveness of separation logic for pointer programs
(3) Completeness of that under deterministic semantics

Ideas:
- Relative completeness and expressiveness for while programs
- A formula that exactly describes the current heap