A Theory of Non-Monotone Memory (Or: Contexts for free)

Eijiro Sumii
Tohoku University

Executive Summary

A sound and complete proof method for arbitrary "contextual" properties

- including contextual equivalence, space improvement, and memory safety
- based on environmental bisimulations [Sumii-Pierce 04, 05] [Koutavas-Wand 06] [Sangiorgi-Kobayashi-Sumii 07]
- for untyped λ-calculus with references and deallocation (`free`)
  - hard to deal with by other methods

Motivation: An Example

Typical implementation of integer multisets (e.g. by linked lists):

set = let r = ref nil in (add_r, mem_r, del_r)

where

add_r = λx. ...insert x into !r...

mem_r = λx. ...search !r for x...

del_r = λx. ...search !r for x and <u>remove</u> the node containing it...

Motivation: Questions

set = let r = ref nil in (add_r, mem_r, del_r)

Is this implementation:

- memory safe?
- observationally equivalent to another implementation (e.g. by binary trees)?
- more (time- or space-) efficient than another implementation?

etc.

Motivation: The Observation

set = let r = ref nil in (add_r, mem_r, del_r)

- It makes no sense to consider the triple (add_r, mem_r, del_r) by itself!
  - because it does nothing (no good, no harm) by itself
- Rather, we must put it under arbitrary contexts C

Motivation: The Problem

How to prove

- memory safety,
- observational equivalence,
- time/space improvement, etc.

under arbitrary contexts C?
(There are infinitely many of them!)

- Naive induction on C does not work
- Traditional logical relations have difficulties with deallocation and with untyped languages (or recursive types)

Our Solution: Generalize Environmental Bisimulations

Represent the states of a context and programs by a set X of tuples $(R, s \triangleright M, s' \triangleright M')$ and (R, s, s')

- Environment R is a binary relation on values, representing the knowledge of the context about the programs
- State $s \triangleright M$ (resp. $s' \triangleright M'$) is a pair of a store and a term, representing the program running on the left (resp. right) hand side of the relation
  - M and M' are omitted when the programs have stopped and are not running

How to use it and how it works

M and M' satisfy P under arbitrary contexts if we can construct some X such that:

- $(\emptyset,\ \emptyset \rhd \lambda x_1 \ldots x_n.M,\ \emptyset \rhd \lambda x_1 \ldots x_n.M') \in X$ for the programs M and M' in question
  - $\{x_1, \ldots, x_n\} \supseteq fv(M, M')$
- X is preserved by reduction of the programs and <u>operations from the context</u>
- Every element in X satisfies the property P in question

The rest of the talk

- The calculus
  - Syntax
  - Operational semantics
- The environmental relations
  - Definition
  - Soundness and completeness
  - Example
  - Unary case
- Conclusion

The calculus: Syntax

Standard untyped, call-by-value λ-calculus with (first-class, higher-order) references and deallocation

M	$::=$	$\ldots$	(standard λ-terms)
		ℓ	(location)
		$\mathsf{ref}\ M$	(allocation)
		$!M$	(dereference)
		$M_1 := M_2$	(update)
		$\mathsf{free}\ M$	(deallocation)
		$M_1 == M_2$	(pointer equality)

The calculus: Operational semantics

Standard small-step reduction with evaluation contexts and stores

$s \triangleright \mathsf{ref}\ V \ \rightarrow\ s \uplus \{\ell \mapsto V\} \triangleright \ell$

$s \triangleright\ !\ell \ \rightarrow\ s \triangleright s(\ell)$

$s \uplus \{\ell \mapsto V\} \triangleright \ell := W \ \rightarrow\ s \uplus \{\ell \mapsto W\} \triangleright ()$

$s \uplus \{\ell \mapsto V\} \triangleright \mathsf{free}\ \ell \ \rightarrow\ s \triangleright ()$

$s \triangleright \ell == \ell \ \rightarrow\ s \triangleright \mathsf{true}$

$s \triangleright \ell_1 == \ell_2 \ \rightarrow\ s \triangleright \mathsf{false}\ \ \text{if } \ell_1 \neq \ell_2$

...and other standard rules...

Caution

Because of deallocation, our reduction is non-deterministic even modulo renaming of fresh locations

$\{\ell \mapsto V\} \rhd \mathsf{free}(\ell); (\mathsf{ref}\ W == \ell)$

$\rightarrow\ \emptyset \rhd \mathsf{ref}\ W == \ell$

$\rightarrow\ \{\ell \mapsto W\} \rhd \ell == \ell$

(or $\{\ell' \mapsto W\} \rhd \ell' == \ell$)

$\rightarrow\ \{\ell \mapsto W\} \rhd \mathsf{true}$

(or $\{\ell' \mapsto W\} \rhd \mathsf{false}$)

The rest of the talk

- The calculus
  - Syntax
  - Reduction
- The environmental relations
  - Definition
  - Soundness and completeness
  - Example
  - Unary case
- Conclusion

The environmental relations: Definition (1/4)

Consider a predicate P on tuples of the forms $(R, s \rhd M, s' \rhd M')$ and (R, s, s')

- Recall: R is the context's knowledge and $s \rhd M$ and $s' \rhd M'$ are the programs' states

- Observational equivalence: $s \rhd M \Uparrow$ iff $s' \rhd M' \Uparrow$
- Space improvement: $|dom(s)| \leq |dom(s')|$

etc.

$X \subseteq P$ is an <u>environmental P-simulation</u> if it is <u>reduction-closed</u> and "<u>operation-closed</u>"

The environmental relations: Definition (2/4)

X is <u>reduction-closed</u> if
for any $(R, s \rhd M, s' \rhd M') \in X$,

- If $s \rhd M \to t \rhd N$, then $s' \rhd M' \to \ldots \to t' \rhd N'$ and $(R, t \rhd N, t' \rhd N') \in X$ (and vice versa)
  - X is preserved by execution of the programs
- If $M = V$ and $M' = V'$, then $(R \cup \{(V,V')\}, s, s') \in X$
  - Context learns values returned by the programs

The environmental relations: Definition (3/4)

X is "<u>operation-closed</u>" if for any $(R, s, s') \in X$,

- For any $(\ell, \ell') \in R$,
  - $(R \cup \{(s(\ell), s'(\ell'))\}, s, s') \in X$ (dereference)
  - $(R, s[\ell \mapsto V], s'[\ell' \mapsto V']) \in X$ (update)
  - $(R, s \backslash \ell, s' \backslash \ell') \in X$ (deallocation)

  where V and V' are arbitrary values composed from R by the context

  - Formally, $(V, V') \in R^* = \{ (C[W_1,\ldots,W_n], C[W_1',\ldots,W_n']) \mid (W_i, W'_i) \in R \}$
- ...continuted...

The environmental relations: Definition (4/4)

X is "operation-closed" if for any $(R, s, s') \in X$,

- For any $(\ell, \ell') \in R$, ...
- For any fresh ℓ and $(V, V') \in R^*$,
  $(R \cup \{(\ell,\ell)\},\ s \uplus \{\ell \mapsto V\},\ s' \uplus \{\ell \mapsto V'\}) \in X$
  (allocation)
- For any $(\lambda x.M, \lambda x.M') \in R$ and $(V, V') \in R^*$,
  $(R,\ s \triangleright (\lambda x.M)V,\ s' \triangleright (\lambda x.M')V') \in X$
  (application)
- For any $((V_1,\dots,V_n), (V_1',\dots,V_n')) \in R$,
  $(R \cup \{(V_i,V_i')\}, s, s') \in X$ (projection)

The rest of the talk

- The calculus
  - Syntax
  - Reduction
- The environmental relations
  - Definition
  - Soundness and completeness
  - Example
  - Unary case
- Conclusion

The environmental relations: Soundness and completeness

Theorem:

The largest P-simulation coincides with the reduction- and context-closure of P

I.e., if programs belong to some P-simulation, then they satisfy P throughout execution of the programs under arbitrary contexts, <u>and vice versa</u>

The rest of the talk

- The calculus
  - Syntax
  - Reduction
- The environmental relations
  - Definition
  - Soundness and completeness
  - Example
  - Unary case
- Conclusion

The environmental relations: Example

- Let set and set' be implementations of integer multisets by linked lists and binary trees
- Then we can prove that set and set'
  - are contextually equivalent, by taking $P(R, s \triangleright M, s' \triangleright M')$ to be $s \triangleright M\Uparrow \Leftrightarrow s' \triangleright M'\Uparrow$
  - use the same number of locations, by taking $P(R, s \triangleright M, s' \triangleright M')$ and $P(R, s, s')$ to be $|dom(s)| = |dom(s')|$

  etc., <u>under arbitrary contexts</u>

$X =$

$\{ (\emptyset, \emptyset \triangleright \lambda\_.set, \emptyset \triangleright \lambda\_.set') \} \cup$

$\{ (R, s \uplus \{\ell \mapsto V\}, s' \uplus \{\ell' \mapsto V'\}) \mid$

$R = \{ (add_{\ell}, add'_{\ell'}), (mem_{\ell}, mem'_{\ell'}), (del_{\ell}, del'_{\ell'}) \}$,
the linked list V and the binary tree V'
respectively represent the same set
under the stores s and s' $\} \cup$

{ ...intermediate states during
insertion/membership/deletion operations... }

⇒ X is an environmental P-simulation
for each P in the previous slide
(up-to context and "up-to allocation")

The rest of the talk

- The calculus
  - Syntax
  - Reduction
- The environmental relations
  - Definition
  - Soundness and completeness
  - Example
  - Unary case
- Conclusion

The environmental relations: Degeneracy to unary case

All of these arguments apply to unary relations (predicates) as well

- In fact, the unary case is degenerated from the binary case, by requiring the left hand side be <u>equal</u> to the right
  - "Simulation" between M and M itself

E.g., Take $P(R, s \rhd M, s' \rhd M')$ to be true iff $s = s'$, $M = M'$ and M is not accessing locations not in dom(s) (memory safety)

Conclusion

- We have developed a proof method
  - for arbitrary contextual properties
  - in untyped λ-calculus
  - with "full" (unrestricted) references
  - and deallocation

In the paper (with online appendices):

- Auxiliary "up-to" techniques
- More examples