A Step-indexed Kripke Model of Hidden State via Recursive Properties on Recursively Defined Metric Spaces

Jan Schwinghammer

Programming Systems Lab
Saarland University

joint work with Lars Birkedal and Kristian Støvring

Talk outline

1. Motivation:
   hiding state
2. Logic-based hiding:
   capabilities, frame and anti-frame rules
3. Possible worlds semantics for logic-based hiding:
   a refined domain equation
   solution in ultrametric spaces

Hidden state

Hidden state is a key design principle used by programmers:

An object (or module, or procedure)

- maintains an internal, mutable data structure,
- its lifetime spans multiple invocations,
- its existence is not revealed in the object's interface description.

Hidden state

For instance, there's hidden state in a memory manager module:

- the module maintains a list of free'd memory chunks, and
- clients only need to know that they obtain "unused" chunks.

Or, in a procedure that uses memoization:

- internal use of a hash table to cache previous calls,
- clients don't depend on the hash table's existence, and how it evolves.

Why hide state?

Hiding state has several benefits for (informal) reasoning.

1. simpler specification of the object:
   specification does not involve the invariant,
2. simpler reasoning about clients:
   no need to thread the object's invariant through client code,
3. less restricted use of the object:
   avoids the need to track aliasing in certain cases.

There should be similar advantages in formal reasoning.

Hiding state in a program logic

The logic-based approach to information hiding

- keeps standard semantics of the programming language
- extends program logic with special proof rules

Hiding state in a program logic

The logic-based approach to information hiding

- keeps standard semantics of the programming language

  here: lambda calculus with state,

  standard operational semantics $(t|h) \longmapsto (t'|h')$
- extends program logic with special proof rules

Hiding state in a program logic

The logic-based approach to information hiding

- keeps standard semantics of the programming language

  here: lambda calculus with state,
  standard operational semantics $(t|h) \longmapsto (t'|h')$
- extends program logic with special proof rules

  here: Charguéraud and Pottier's type and capability system,
  frame and anti-frame rules [ICFP'08; LICS'08]

Charguéraud and Pottier's types and capabilities

Capabilities describe heaps:

$$C ::= \mathbf{emp} \mid \{\sigma : \tau\} \mid C_1 * C_2 \mid \ldots$$

For instance, $\{\sigma_1 : \text{ref int}\} * \{\sigma_2 : \text{ref int}\}$

Types describe values:

$$\tau ::= \text{int} \mid [\sigma] \mid \underbrace{\tau_1 * C_1}_{\chi_1} \to \underbrace{\tau_2 * C_2}_{\chi_2} \mid \ldots$$

For instance, $\mathit{deref} : [\sigma] * \{\sigma : \text{ref } \tau\} \to \tau * \{\sigma : \text{ref } \tau\}$

Extension of specifications by invariants

A type-theoretic connective expresses invariant extension:

a specification *invariant*

$$\underbrace{\tau \otimes C}$$

modified specification:

pre- and postconditions of all arrows in τ extended by $*C$

Extension of specifications by invariants

A type-theoretic connective expresses invariant extension:

a specification *invariant*

$$\underbrace{\tau \otimes C}$$

modified specification:

pre- and postconditions of all arrows in τ extended by $* C$

Formally expressed by a type equivalence:

- $(\chi_1 \to \chi_2) \otimes C \quad \equiv \quad (\chi_1 \otimes C) * C \to (\chi_2 \otimes C) * C$
- $(\tau \otimes C) \otimes C' \quad \equiv \quad \tau \otimes ((C \otimes C') * C')$
- $\{\sigma : \tau\} \otimes C \quad \equiv \quad \{\sigma : \tau \otimes C\}$

Hiding state with frame and anti-frame rules

Shallow Frame
cf. Separation logic

$$\frac{\Vdash t : \chi_1 \to \chi_2}{\Vdash t : \chi_1 * C \to \chi_2 * C}$$

Deep Frame
Schwinghammer et al., CSL'09

$$\frac{\Vdash t : \chi_1 \to \chi_2}{\Vdash t : (\chi_1 \otimes C) * C \to (\chi_2 \otimes C) * C}$$

Anti-frame
Pottier, LICS'08

$$\frac{\Vdash t : (\tau \otimes C) * C}{\Vdash t : \tau}$$

Explicating quantification over invariants

Intuition:

- Rules exploit implicit quantification over invariants.
- The semantics of arrow types makes quantification explicit:

$$\underbrace{\Vdash t : \chi_1 \to \chi_2}_{\textit{our interpretation}} \quad \text{if} \quad \underbrace{\Vdash t : \forall C.\, \chi_1 \circ C \to \exists C'.\, \chi_2 \circ (C \circ C')}_{\textit{standard interpretation}}$$

where $\cdot \circ C \stackrel{def}{=} (\cdot \otimes C) * C$.

Invariants as possible worlds

In the semantics,

- invariants C form set of worlds W,
- capabilities and types depend on these worlds,

$$Cap \stackrel{def}{=} W \to \mathcal{P}(Heap) \qquad Type \stackrel{def}{=} W \to \mathcal{P}(Val) \ ,$$

- invariants are arbitrary capabilities,

$$W \ \cong \ Cap \ .$$

Technicalities, 1

Uniform predicates $p \subseteq \mathbb{N} \times \mathit{Heap}$ as metric space

uniformity	$(n,h) \in p \ \wedge\ j \leq n \ \Rightarrow\ (j,h) \in p$
approximation	$p_{[n]} = \{(k,h) \in p \mid k < n\}$
distance	$d(p,q) = \inf\{2^{-n} \mid p_{[n]} = q_{[n]}\}$

Theorem (America & Rutten, 1989)

There exists a unique $W \in \mathit{CBUlt}$ such that

$$W \ \cong\ 1/2 \cdot W \to \mathit{UPred}(\mathit{Heap})$$

Monotonicity

Requirement:
Hidden state of non-local objects must not invalidate specifications.

Composition. Invariants can be combined:

$$\text{composition operation} \qquad (c \circ c')(w) \stackrel{def}{=} (c \otimes c')(w) * c'(w)$$

$$\text{invariant extension} \qquad (c \otimes c')(w) \stackrel{def}{=} c(c' \circ w)$$

Kripke monotonicity.
$w \circ w'$ is a "future world" of w: $w \leq w \circ w'$,
and capabilities need to satisfy:

$$\text{monotonicity} \qquad w_1 \leq w_2 \;\Rightarrow\; c(w_1) \subseteq c(w_2)$$

Technicalities, 2

In summary, we are looking for a solution

$$\hat{W} \;\cong\; 1/2 \cdot \hat{W} \to_{mon} UPred(Heap)$$

The definition of the order on $\hat{W}$ uses this isomorphism:

$$w_1 \leq w_2 \;\overset{def}{\Leftrightarrow}\; \exists w.\; w_2 = w_1 \circ w$$

$$(w_1 \circ w_2)(w) \;=\; w_1(\underbrace{w_2 \circ w}_{\textit{world}}) * \underbrace{w_2(w)}_{\textit{capability}}$$

Consequence:
Standard existence theorems like America & Rutten's do not apply.
Previously: tedious inverse limit construction in *CBUlt* [FOSSACS'10].

Our approach: hereditarily monotonic worlds

Theorem (Hereditarily monotonic worlds)

There exists $\hat{W} \subseteq W$ such that

$$c \in \hat{W} \;\Leftrightarrow\; \forall w_1, w_2 \in \hat{W}.\; c(w_1) \subseteq c(w_1 \circ w_2)$$

Proof idea:

- Consider the set *Rel* of non-empty closed relations $R \subseteq W$
- $Rel \in CBUlt$, when equipped with Hausdorff distance
- $\hat{W}$ is the fixed point of contractive function $\Phi : Rel \to Rel$

Connecting the dots

Define a step-indexed semantics of types: arrow types

$$(k, \lambda x.t) \in [\![\chi_1 \to \chi_2]\!](w)$$

if and only if

$$\begin{aligned}
&\forall j < k.\ \forall w' \in \hat{W}.\\
&(j, (v, h)) \in [\![\chi_1]\!](w \circ w') * \iota(w \circ w')(emp)\\
&\wedge\ (t[x{:=}v]|h) \longmapsto^{i} (t'|h') \not\longmapsto\\
&\Rightarrow\ \exists w'' \in \hat{W}.\ (j{-}i, (t', h')) \in [\![\chi_2]\!](w{\circ}w'{\circ}w'') * \iota(w{\circ}w'{\circ}w'')(emp)
\end{aligned}$$

Key ideas:

- universal and existential quantification over worlds
- using worlds as invariants
- linking uniformity and operational semantics

Summary

Frame and anti-frame rules formalize reasoning about hidden state

- specifications are "parametric" in non-local invariants
- possible-worlds model with recursive worlds
- hereditarily monotonic functions, constructed in two steps

Technically, a combination of operational and denotational ideas

- uniform predicates from step-indexing [Appel & McAllester, 2001]
- recursive metric spaces [America & Rutten, 1989]
- recursive predicates via Banach fixpoint theorem

Outlook

Study hidden state in richer programming languages

- continuations
- concurrency

Study Pottier's generalized frame and anti-frame rules

- evolving "invariants"
- parametrized recursive worlds

Thank you.