Separation Logic and the Mashup Isolation Problem

Ankur Taly

Dept. of Computer Science, Stanford University

Phd Qualifier Exam Talk

Outline

1. Background
   - Hoare Logic
   - Intuition behind Separation Logic
2. The Mashup Isolation problem ?
   - Formal Definition of Mashups
   - Isolation Property
   - How does Separation Logic help ?
3. Basic Separation Logic
   - Assertion language and Inference rules
   - Solving the Isolation Problem
4. Separation Logic with Permissions
   - Assertion language and Inference rules
   - Solving the Isolation Problem
5. Ongoing and Future Work

While Programs

$$
\begin{array}{lcl}
\textit{Expressions } E & : & x \mid n \mid E + E \mid E * E \\
\textit{Boolean Expressions } B & : & \textit{true} \mid \textit{false} \mid E = E \mid B \implies B \\
\textit{Commands } C, D & : & x := E \mid \textit{if } B \textit{ then } C \textit{ else } C \mid \\
 & & \textit{while } B \textit{ then } C \mid C; C
\end{array}
$$

Store

$Vars$: Set of Variables $\quad$ $StoreValues$: Nat

$Stores\ A, B : Vars \rightharpoonup StoreValues$

Each program C evaluates with respect to store: $A_1, C_1 \rightarrow A_2, C_2$

Hoare Logic

- *Axiomatic method for proving properties of while-programs, invented by Hoare in 1969.*
- **Central Idea:** Assign meaning to a program C using a *Hoare triple* $\{P\}C\{Q\}$
  - P : Assertion on variables *before* C begins execution.
  - Q: Assertion on variables *after* C finishes execution
- Example:
  - $\{x = 10\}y = x + 1\{y = 11\}$.
  - $\{x = y\}$*while* $x = 10$ *then* $x = x + 1; y = y + 1\{x = y\}$.

Validity of triples

A Hoare triple $\{P\}C\{Q\}$ is valid IFF:
IF P holds initially and C terminates THEN Q holds finally.

The Inference System

The assertions P, Q in the triple $\{P\}C\{Q\}$ are predicates on the store.

$$\{P[E/x]\}x := E\{P\} \qquad [\text{S-ASSIGNMENT}]$$

$$\frac{\{P\}C_1\{P'\} \quad \{P'\}C_2\{Q\}}{\{P\}C_1; C_2\{Q\}} \qquad [\text{S-SEQ}]$$

$$\frac{\{P \wedge B\}C_1\{Q\} \quad \{P \wedge \neg B\}C_2\{Q\}}{\{P\}\mathit{if}\ B\ \mathit{then}\ C_1\ \mathit{else}\ C_2\{Q\}} \qquad [\text{S-IF}]$$

$$\frac{\models P \implies P' \quad \{P'\}C\{Q'\} \quad \models Q' \implies Q}{\{P\}C\{Q\}} \qquad [\text{S-CONSEQ}]$$

- Notice the simplicity of the assignment axiom.
- Assignment axiom also provides Weakest-pre-condition.

$\mathcal{L}$: While-programs + references and records

Introduce Heaps: Mappings from locations to records

- Variables can be locations or numbers.
- Two new boolean expressions: $isNat?(x), isLoc?(x)$.
- Three new commands:
  - $x.p := E$ Update property p of record at location x.
  - $x_1 := x_2.p$ Lookup property p of records at location x_2.
  - $x := \{p_i : E_i\}_{i \in \{1,\ldots,n\}}$ Record Creation.

Heaps and Stores

Loc : Set of locations $\qquad$ $\mathbb{P}$: Set of Property names

$StoreValues : Loc \cup Nat$

Stores $A, B : Vars \rightharpoonup StoreValues$

Heaps $H, C : Loc \rightharpoonup \mathbb{P} \rightharpoonup StoreValues$

Each program C evaluates with respect to a heap and a store.

Hoare logic for $\mathcal{L}$

- We need assertions on heap-stores now: $cont\ x.p = E$
  - Meaning: Property p of record at location x has value of expression E.
  - Think of $cont$ as the function $Loc \rightharpoonup \mathbb{P} \rightharpoonup StoreValues$.
- Is $\{cont\ y\ p = 10\}x.p = 11\{cont\ y\ p = 10\}$ valid ?
  - No, x and y may contain same location.
  - Rule of Constancy does not hold.
- Correct Triple: $\{cont_{x.p:=11}\, y.p = 10\}x.p = 11\{cont\ y\ p = 10\}$
  - $cont_{x.p:=11} = \lambda l, q : \mathit{If}\ (l = x)\ \wedge\ (p = q)\ \mathit{then}\ 10\ \mathit{else}\ cont\ l\ q$
  - This is too complex, imagine multiple assignments to $x.p$.
- Intuitively, $cont\ y\ p = E$ is preserved during execution of C if y is disjoint from location-properties touched by C.
- Can I prove that $cont\ y\ p = E$ is preserved without threading it through the entire analysis of C ?

This is where Separation Logic comes in !

Hoare logic for $\mathcal{L}$

- We need assertions on heap-stores now: *cont* $x.p = E$
  - Meaning: Property p of record at location x has value of expression E.
  - Think of *cont* as the function $Loc \rightharpoonup \mathbb{P} \rightharpoonup StoreValues$.
- Is $\{cont\ y\ p = 10\}x.p = 11\{cont\ y\ p = 10\}$ valid ?
  - No, x and y may contain same location.
  - Rule of Constancy does not hold.
- Correct Triple: $\{cont_{x.p:=11} y.p = 10\}x.p = 11\{cont\ y\ p = 10\}$
  - $cont_{x.p:=11} = \lambda l, q : If\ (l = x) \wedge (p = q)\ then\ 10\ else\ cont\ l\ q$
  - This is too complex, imagine multiple assignments to $x.p$.
- Intuitively, $cont\ y\ p = E$ is preserved during execution of C if y is disjoint from location-properties touched by C.
- Can I prove that $cont\ y\ p = E$ is preserved without threading it through the entire analysis of C ?

This is where Separation Logic comes in !.

Hoare logic for $\mathcal{L}$

- We need assertions on heap-stores now: $cont\ x.p = E$
  - Meaning: Property p of record at location x has value of expression E.
  - Think of $cont$ as the function $Loc \rightharpoonup \mathbb{P} \rightharpoonup StoreValues$.
- Is $\{cont\ y\ p = 10\}x.p = 11\{cont\ y\ p = 10\}$ valid ?
  - No, x and y may contain same location.
  - Rule of Constancy does not hold.
- Correct Triple: $\{cont_{x.p:=11} y.p = 10\}x.p = 11\{cont\ y\ p = 10\}$
  - $cont_{x.p:=11} = \lambda l, q : \mathit{If}\ (l = x)\ \wedge\ (p = q)\ \mathit{then}\ 10\ \mathit{else}\ cont\ l\ q$
  - This is too complex, imagine multiple assignments to $x.p$.
- Intuitively, $cont\ y\ p = E$ is preserved during execution of C if y is disjoint from location-properties touched by C.
- Can I prove that $cont\ y\ p = E$ is preserved without threading it through the entire analysis of C ?

This is where Separation Logic comes in !.

Separation Logic

History:

1. *Burstall, 1972*: *separate program texts which work on separated sections of the store can be reasoned about independently*.
2. *Reynolds, MPC 2000*: An Intuitionistic logic based on Burstall's observation. Introduced $*$.
3. *Ishtiaq and O'Hearn, POPL 2001*: A classical version of Reynold's logic. Introduced $-\!\!*$
4. *Reynolds, LICS 2002*: Generalized the above logic to arbitrary pointer arithmatic.
5. Several variants for specific domains have been discovered:
   - *O'Hearn et al, POPL 2004*: Separation and Information hiding.
   - *Bornat et al, POPL 2005*: Separation logic with permissions.
   - Many more ...

Basic Separation Logic

Change of Notation: $x.p \mapsto E$ instead of *cont* x p E.

- **Separating conjunction** $*$: $(x.p \mapsto 10) * (y.p \mapsto 10)$ means
  - Field p of locations x and y, contains 10.
  - x and y are different locations.
  - Therefore $\{x.p \mapsto 10 * y.p \mapsto 10\}x.p := 11\{x.p \mapsto 11 * y.p \mapsto 10\}$ is valid.
- **Local Reasoning:** A specification should *only* reason about the heap locations and variables accessed.
  - Assignment axiom for $x.p := E$ is simply $\{x.p \mapsto \_\}x.p := E\{x.p \mapsto E\}$
  - How do we derive that $y.p \mapsto 10$ is preserved under the assignment $x.p := 11$ if $x \neq y$?
- **Frame Rule:** $\frac{\{P\}C\{Q\}}{\{P*R\}C\{Q*R\}}$
  - Helps in going from local specifications to global specifications.
  - Soundness of frame rule $\implies$ *A specified program never looks beyond what is present in its pre-condition !*

Basic Separation Logic

Change of Notation: $x.p \mapsto E$ instead of *cont* x p E.

- **Separating conjunction** $*$: $(x.p \mapsto 10) * (y.p \mapsto 10)$ means
  - Field p of locations x and y, contains 10.
  - x and y are different locations.
  - Therefore $\{x.p \mapsto 10 * y.p \mapsto 10\}x.p := 11\{x.p \mapsto 11 * y.p \mapsto 10\}$ is valid.
- **Local Reasoning:** A specification should *only* reason about the heap locations and variables accessed.
  - Assignment axiom for $x.p := E$ is simply $\{x.p \mapsto \_\}x.p := E\{x.p \mapsto E\}$
  - How do we derive that $y.p \mapsto 10$ is preserved under the assignment $x.p := 11$ if $x \neq y$?
- **Frame Rule:** $\frac{\{P\}C\{Q\}}{\{P*R\}C\{Q*R\}}$
  - Helps in going from local specifications to global specifications.
  - Soundness of frame rule $\implies$ *A specified program never looks beyond what is present in its pre-condition !*

Basic Separation Logic

Change of Notation: $x.p \mapsto E$ instead of *cont* x p E.

- **Separating conjunction** $*$: $(x.p \mapsto 10) * (y.p \mapsto 10)$ means
  - Field p of locations x and y, contains 10.
  - x and y are different locations.
  - Therefore $\{x.p \mapsto 10 * y.p \mapsto 10\} x.p := 11 \{x.p \mapsto 11 * y.p \mapsto 10\}$ is valid.
- **Local Reasoning:** A specification should *only* reason about the heap locations and variables accessed.
  - Assignment axiom for $x.p := E$ is simply $\{x.p \mapsto \_\} x.p := E \{x.p \mapsto E\}$
  - How do we derive that $y.p \mapsto 10$ is preserved under the assignment $x.p := 11$ if $x \neq y$?
- **Frame Rule:** $\dfrac{\{P\}C\{Q\}}{\{P*R\}C\{Q*R\}}$
  - Helps in going from local specifications to global specifications.
  - Soundness of frame rule $\implies$ *A specified program never looks beyond what is present in its pre-condition !*

Run-time Safety

- *Robin Milner*: *Well specified programs never go wrong*.
- Any specification $\{P\}C\{Q\}$ is such that for any heap-store H, A which satisfies P, executing C on H, A will never lead to a run-time error. This means:
  - All locations and variables accessed are mentioned by P so that there are no memory errors (critical for frame rule).
  - P includes sufficient conditions on the values of variables so that there are no type errors.
- Therefore a specification $\{x.p \mapsto 5\}C\{x. \mapsto 10\}$ means that C only accesses property p of location contained in x.
- This was the main motivation behind using separation logic for proving mashup isolation.

Separation Logic with Permissions

- Invented by Bornat et al in 2005 by incorporating fractional permissions in Separation logic.
- Main motivation was to track permissions in threads.
- We consider it in a sequential setting with 3 permissions - $\{r\}, \{w\}, \{r, w\}$.

Key Ideas:

- Add more information in the specification to reason about what is read-only.
- Read $x.p \mapsto v$ as "program has permission to read/write property p of record at x".
- Three new assertions: $x.p \overset{\{r\}}{\mapsto} E$, $x.p \overset{\{w\}}{\mapsto} E$, $x.p \overset{\{r,w\}}{\mapsto} E$.
- Modified $*$: $x.p \overset{r}{\mapsto} 10 * y.p \overset{r}{\mapsto} 10$ can hold even if $x = y$.
- Frame rule stays the same !

Outline

1. Background
   - Hoare Logic
   - Intuition behind Separation Logic
2. The Mashup Isolation problem ?
   - Formal Definition of Mashups
   - Isolation Property
   - How does Separation Logic help ?
3. Basic Separation Logic
   - Assertion language and Inference rules
   - Solving the Isolation Problem
4. Separation Logic with Permissions
   - Assertion language and Inference rules
   - Solving the Isolation Problem
5. Ongoing and Future Work

Web 2.0

All about mixing and merging content (data and code) from multiple content providers in the users browser, to provide high-value applications known as mashups

- Notation:
  - Individual contents being mixed - *Components*.
  - Content Providers - *Principals*.
  - Publisher of the mashup- *Host*.
- Examples:
  - Basic Mashup: Any web page with advertisements, Facebook page with applications
  - More complex mashups: Yelp, Yahoo Newsglobe ...

Security Issues in Mashups

- Principals participating in a mashup are usually mutually untrusting.
- Each component must be protected from malicious behavior of other components.
  - Each Facebook application wants to make sure that its variables are not over-written by other applications.
  - Current *FBJS* mechanism not sufficient for ensuring this.

This Work:

- Focus on non-interacting basic mashups.
- Verify complete inter-component isolation.

Our Model

Basic Mashups (defined first in our Oakland 2010 paper):

- Components are programs in some sequential prog. language
- Mashup is a sequential composition of the components after variable renaming: $Rn(C_1); \ldots ; Rn(C_n)$.
- Reasonable model for a web page with multiple advertisements.

Isolation Property: *Behavior of each component as part of the mashup should be similar to the behavior obtained by executing it independently.*

- Isolation property in Oakland paper is a special case of the above.
- This work focusses on verifying the property whereas the Oakland paper focusses on enforcing it.

Prog. Language: Simple Imperative language with references and records. Far from *JavaScript*, but good starting point for testing out new theoretical techniques.

Our Model

Basic Mashups (defined first in our Oakland 2010 paper):

- Components are programs in some sequential prog. language
- Mashup is a sequential composition of the components after variable renaming: $Rn(C_1); \ldots; Rn(C_n)$.
- Reasonable model for a web page with multiple advertisements.

Isolation Property: *Behavior of each component as part of the mashup should be similar to the behavior obtained by executing it independently.*

- Isolation property in Oakland paper is a special case of the above.
- This work focusses on verifying the property whereas the Oakland paper focusses on enforcing it.

Prog. Language: Simple Imperative language with references and records. Far from *JavaScript*, but good starting point for testing out new theoretical techniques.

Formal Definition of Mashups

- Principals: $id_1, \ldots, id_n$.
- Components $(C_1, id_1), \ldots, (C_n, id_n)$: Programs from $\mathcal{L}$.
- Initial execution environment: Heap-store H, A
- $Rn(C, a)$: Command obtained by replacing all $x \in C$ with $a.x$.
- $Rn(A, a)$: Store obtained by replacing all x in A with $a.x$.

Variable-separated mashup

A *variable-separated mashup* $\mathcal{M}(H, A, (C_1, id_1), \ldots, (C_n, id_n))$ is defined as the state $H_{mash}, A_{mash}, D_1; \ldots; D_n$ where

- $H_{mash} := H$.
- $A_{mash} := Rn(A, id_1). \ldots .Rn(A, id_n)$.
- $D_i := Rn(C_i, id_i)$.

Variable renaming is done so that components cannot influence each other via the store.

Formal Definition of Mashups

- Principals: $id_1, \ldots, id_n$.
- Components $(C_1, id_1), \ldots, (C_n, id_n)$: Programs from $\mathcal{L}$.
- Initial execution environment: Heap-store H, A
- $Rn(C, a)$: Command obtained by replacing all $x \in C$ with $a.x$.
- $Rn(A, a)$: Store obtained by replacing all x in A with $a.x$.

Variable-separated mashup

A *variable-separated mashup* $\mathcal{M}(H, A, (C_1, id_1), \ldots, (C_n, id_n))$ is defined as the state $H_{mash}, A_{mash}, D_1; \ldots; D_n$ where

- $H_{mash} := H$.
- $A_{mash} := Rn(A, id_1). \ldots .Rn(A, id_n)$.
- $D_i := Rn(C_i, id_i)$.

Variable renaming is done so that components cannot influence each other via the store.

Operational Semantics of $\mathcal{L}$

- Expressions $[\![E]\!]_{Exp} : Stores \rightarrow StoreValues \cup \{error\}$.
- Boolean Expr $[\![B]\!]_{Bexp} : Stores \rightarrow \{true, false, error\}$.
- Program states S, T are formalized as triples (H, A, C)
- Commands: $\dfrac{\langle premise \rangle}{H_1, A_1, C_1 \rightarrow H_2, A_2, C_2}$(small step)

Example rules:

$$\frac{H, A, C_1 \longrightarrow K, B, C_1'}{H, A, C_1; C_2 \longrightarrow K, B, C_1'; C_2} \quad [\text{C-SequenceContinue}]$$

$$\frac{l \in dom(H) \;\; AND \;\; x \in dom(A)}{H, A, x := l.p \rightarrow H, A[x \rightarrow H(l).p], normal} \quad [\text{C-LookupNormal}]$$

$$\frac{l \notin dom(H) \;\; OR \;\; x \notin dom(A)}{H, A, x := l.p \rightarrow H, A, abort} \quad [\text{C-LookupAbort}]$$

Notations and Definitions

$dom(H)$: $\{(l, p) \mid H(l)(p) \text{ is defined}\}$.
$dom(A)$: $\{x \mid A(x) \text{ is defined}\}$.
Given states S, T:

- $\mathcal{H}(S), \mathcal{S}(S), \mathcal{C}(S)$ denote the heap store and term part of the trace.
- $S \rightsquigarrow T$: S goes to T in zero or more steps.
- $Traces(S)$: Set of reduction traces of S.
- $S\uparrow \quad \stackrel{def}{=} \neg\exists T : S \rightsquigarrow T \not\rightarrow$.
- $Safe(S) \quad \stackrel{def}{=} \forall T : S \rightsquigarrow T \implies \mathcal{C}(T) \neq abort$.

The Simulation Relation

Heap Actions: (l, p, a, v) where $a \in \{r, w\}$ and $v \in StoreValues$
Store Actions: (x, a, v) where $a \in \{r, w\}$ and $v \in StoreValues$

- Given a trace τ, $Acc(\tau)$ is the action sequence corresponding to the trace

State simulation $S \sim T$

There exists a variable renaming $ren : Vars \rightarrow Vars$:

1. *Safe Monotonicity.* $Safe(S) \implies Safe(T)$
2. *Termination Monotonicity.* $\neg S \uparrow \wedge Safe(S) \implies \neg T \uparrow \wedge Safe(S)$
3. *Access Similarity* If $Safe(S)$ holds then for all $\tau_T \in Traces(T)$, there exists $\tau_S \in Traces(S)$ such that $ren(Acc(\tau_S)) = Acc(\tau_T)$

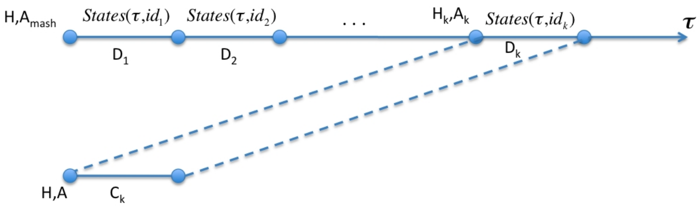
Isolation Property

Let $\mathcal{M}(H, A, (C_1, id_1), \ldots, (C_n, id_n)) = H, A_{mash}, D_1; \ldots ; D_n$, where D_i is a renamed version of C_i.

Pick any $\tau \in Traces(H, A_{mash}, D_1; \ldots ; D_n)$.

- By semantics of sequential compositions, starting heap-store for D_{i+1} is the final heap-store for D_i.
- Define $States(\tau, id_i))$ as the sub-trace of τ corresponding to execution of D_i.
- $States(\tau, id_i)) = \emptyset$ if for some $j < i$, D_j doesn't terminate normally.

Isolation Property, formally



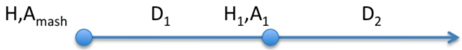
Isolation Property

$\mathcal{M}(H, A, (C_1, id_1), \ldots, (C_n, id_n))$ is isolated IFF:
for all traces $\tau \in Traces(\mathcal{M}((C_1, id_1), \ldots, (C_n, id_n)))$,

$$\forall i : States(\tau, id_i) \neq \emptyset \implies (H, A, C_i) \sim (H_i, A_i, D_i)$$

where $H_i, A_i = \mathcal{HS}(States(\tau, id_i))$

How does Separation Logic help ?



Consider heap store H, A and two components C_1, C_2.
$\mathcal{M}(H, A, (C_1, id_1), (C_2, id_2))) = H, A_{mash}, D_1; D_2$:
Let H_1, A_1 be such that $H, A_{mash}, D_1 \rightsquigarrow H_1, A_1, normal$ (assuming termination).

Result

$Isolation(\mathcal{M}(H, A, (C_1, id_1), (C_2, id_2))))$ IFF:

$$\forall l, p : l, p \in Read(H, A_{mash}, D_2) \cap dom(H) \implies H(l).p = H_1(l).p$$

How does separation logic help

$\forall l, p : l, p \in Read(H, A_{mash}, D_2) \cap dom(H) \implies H(l).p = H_1(l).p$

IFF: one of the following holds:

Any location-property pair that is read during the reduction of D_2 on H, A_{mash} is:

A. Not accessed during the reduction of D_1 on H, A
  - Basic Separation logic can tell me what is accessed and whether it is disjoint from a certain portion of the heap.

B. At most read during the reduction of D_1 on H, A
  - Definition of isolation in Oakland paper.
  - Separation logic with permissions - modified separating conjunction allows overlap on read-only portion.

C. At most written and restored during the reduction of D_1 on H, A
  - Tricky, I have a naive solution.
  - Separation logic and Information hiding.

How does separation logic help

$\forall l, p : l, p \in Read(H, A_{mash}, D_2) \cap dom(H) \implies H(l).p = H_1(l).p$

IFF: one of the following holds:

Any location-property pair that is read during the reduction of D_2 on H, A_{mash} is:

A. Not accessed during the reduction of D_1 on H, A
   - Basic Separation logic can tell me what is accessed and whether it is disjoint from a certain portion of the heap.

B. At most read during the reduction of D_1 on H, A
   - Definition of isolation in Oakland paper.
   - Separation logic with permissions - modified separating conjunction allows overlap on read-only portion.

C. At most written and restored during the reduction of D_1 on H, A
   - Tricky, I have a naive solution.
   - Separation logic and Information hiding.

How does separation logic help

$\forall l, p : l, p \in Read(H, A_{mash}, D_2) \cap dom(H) \implies H(l).p = H_1(l).p$
IFF: one of the following holds:
Any location-property pair that is read during the reduction of D_2 on H, A_{mash} is:

A. Not accessed during the reduction of D_1 on H, A
   - Basic Separation logic can tell me what is accessed and whether it is disjoint from a certain portion of the heap.

B. At most read during the reduction of D_1 on H, A
   - Definition of isolation in Oakland paper.
   - Separation logic with permissions - modified separating conjunction allows overlap on read-only portion.

C. At most written and restored during the reduction of D_1 on H, A
   - Tricky, I have a naive solution.
   - Separation logic and Information hiding.

Outline

1. Background
   - Hoare Logic
   - Intuition behind Separation Logic
2. The Mashup Isolation problem ?
   - Formal Definition of Mashups
   - Isolation Property
   - How does Separation Logic help ?
3. Basic Separation Logic
   - Assertion language and Inference rules
   - Solving the Isolation Problem
4. Separation Logic with Permissions
   - Assertion language and Inference rules
   - Solving the Isolation Problem
5. Ongoing and Future Work

Basic Separation Logic SL_1

Assertion language $\mathcal{A}_1$

$$P := B \mid emp \mid x.p \mapsto E \mid P * P \mid P \mathrel{-\!\!*} P \mid P \implies P \mid \exists x.P$$

Satisfaction of Assertions: $H, A \models_{\mathcal{A}_1} P$

- Boolean Exp: $H, A \models_{\mathcal{A}_1} B \quad iff \quad [\![B]\!]_{Bexp} A = true$
- Points-to: $H, A \models_{\mathcal{A}_1} E_1.p \mapsto E_2$ iff
  1. $dom(H) = \{([\![E_1]\!]_{Exp} A, p)\}$
  2. $H([\![E_1]\!]_{Exp} A).p = [\![E_2]\!]_{Exp} A$
- Notice that the points-to relation is exact.

Satisfaction of Assertions

Separating Conjunction: $H, A \models_{\mathcal{A}_2} P_1 * P_2$ iff
$\exists H_1, H_2$:

- $dom(H_1) \cap dom(H_2) = \emptyset$
- $H_1.H_2 = H$
- $H_1, A \models_{\mathcal{A}_1} P_1 \wedge H_2, A \models_{\mathcal{A}_1} P_2$

Remarks:

- No store separation, we can write $(l_1.p \mapsto x) * (l_2.p \mapsto x)$.
- Semantics is exact.

Empty Heap: $H, A \models_{\mathcal{A}_1} emp \quad iff \quad dom(H) = \emptyset$
Implication: $H, A \models_{\mathcal{A}_1} P_1 \implies P_2 \quad iff$
$H, A \models_{\mathcal{A}_1} P_1 \implies H, A \models_{\mathcal{A}_1} P_2$

Satisfaction of Assertions

We have local specifications and frame rule, but how do we express pre-conditions for an arbitrary assertion P ?

$\{??\}x.p = 10\{P\}$

Separating Implication ($-\!\!*$):

- $x.p \mapsto 10 -\!\!* P$ holds for heaps to which if a heap satisfying $x.p \mapsto 10$ is concatenated then assertion P holds.
- Therefore, $\{x.p \mapsto \_ * (x.p \mapsto 10 -\!\!* P)\}x.p = 10\{P\}$ is valid.

Formally, $H, A \models_{\mathcal{A}_1} P_1 -\!\!* P_2 \quad iff$

$\forall H_1, H_2$:

- $dom(H_1) \cap dom(H) = \emptyset$
- $H.H_1 = H_2$
- $H, A \models_{\mathcal{A}_1} P_1 \wedge H_2, A \models_{\mathcal{A}_1} P_2$

Satisfaction of Assertions

We have local specifications and frame rule, but how do we express pre-conditions for an arbitrary assertion P ?
$\{??\}x.p = 10\{P\}$
Separating Implication ($-\!\!*$):

- $x.p \mapsto 10 -\!\!* P$ holds for heaps to which if a heap satisfying $x.p \mapsto 10$ is concatenated then assertion P holds.
- Therefore, $\{x.p \mapsto \_ * (x.p \mapsto 10 -\!\!* P)\}x.p = 10\{P\}$ is valid.

Formally, $H, A \models_{\mathcal{A}_1} P_1 -\!\!* P_2 \quad iff$
$\forall H_1, H_2$:

- $dom(H_1) \cap dom(H) = \emptyset$
- $H.H_1 = H_2$
- $H, A \models_{\mathcal{A}_1} P_1 \wedge H_2, A \models_{\mathcal{A}_1} P_2$

Satisfaction of Assertions

We have local specifications and frame rule, but how do we express pre-conditions for an arbitrary assertion P ?

$\{??\}x.p = 10\{P\}$

Separating Implication ($-\!\!*$):

- $x.p \mapsto 10 -\!\!* P$ holds for heaps to which if a heap satisfying $x.p \mapsto 10$ is concatenated then assertion P holds.
- Therefore, $\{x.p \mapsto \_ * (x.p \mapsto 10 -\!\!* P)\}x.p = 10\{P\}$ is valid.

Formally, $H, A \models_{\mathcal{A}_1} P_1 -\!\!* P_2 \quad iff$

$\forall H_1, H_2$:

- $dom(H_1) \cap dom(H) = \emptyset$
- $H.H_1 = H_2$
- $H, A \models_{\mathcal{A}_1} P_1 \wedge H_2, A \models_{\mathcal{A}_1} P_2$

Axioms and Inference rules (SL_1)

Hoare logic rules apply. In addition:

$$\{E_1.p \mapsto \_ \wedge Wt(E_2)\} E_1.p = E_2 \{E_1.p \mapsto E_2 \wedge Wt(E_2)\} \qquad []$$

$$\frac{y \text{ not free in } E}{\{present(x) \wedge \exists y : E.p \mapsto y\} x = E.p \{\exists y : E[y/x].p \mapsto x\}} \qquad []$$

$$\frac{y \text{ not free in any } E_i}{\{present(x) \wedge Wt(\tilde{E_i}) \wedge emp\} x := \{\tilde{p_i} : \tilde{E_i}\}\{\exists y : (x : \{p_i \mapsto E_i[y/x]\})\}} \qquad []$$

- "Backwords" rules for arbitrary post-conditions can be written for each of the above commands.
- Ishtiaq and O'Hearn (POPL 2001) proved that the backwords axioms express weakest-pre-conditions.

Frame rule

$$\frac{\{P\}C\{Q\}}{\{P * R\}C\{Q * R\}}[\mathit{modifies}(C) \cap \mathit{free}(R) = \emptyset]$$

$\mathit{modifies}(C)$ can be syntactically derived from C.

Proposition

For H, A and K, B such that $H \subseteq K$ and $A \subseteq B$, for all commands C

1. *Safe Monotonicity.* $Safe(H, A, C) \implies Safe(K, B, C)$
2. *Frame Property.* If $Safe(H, A, C)$ holds then:
   For all K', B', C' such that $K, B, C \rightsquigarrow K', B', C'$, exists H', A'':

$$H, A, C \rightsquigarrow H', A', C' \bigwedge K' - H' = K - H \bigwedge B - A = B' - A'$$

Soundness of frame rule follows from above proposition.

Frame rule

$$\frac{\{P\}C\{Q\}}{\{P * R\}C\{Q * R\}}[modifies(C) \cap free(R) = \emptyset]$$

$modifies(C)$ can be syntactically derived from C.

Proposition

For H, A and K, B such that $H \subseteq K$ and $A \subseteq B$, for all commands C

1. *Safe Monotonicity.* $Safe(H, A, C) \implies Safe(K, B, C)$
2. *Frame Property.* If $Safe(H, A, C)$ holds then:
   For all K', B', C' such that $K, B, C \rightsquigarrow K', B', C'$, exists H', A'':

$$H, A, C \rightsquigarrow H', A', C' \bigwedge K' - H' = K - H \bigwedge B - A = B' - A'$$

Soundness of frame rule follows from above proposition.

Soundness

$$[\![P]\!]_{\mathcal{A}_1} \stackrel{def}{=} \{H, A \mid H, A \models_{\mathcal{A}_1} P\}$$

Validity $\models_{SL_1}$

$\{P\}C\{Q\}$ is SL_1-*valid* IFF: for all heap-stores $H, A \in [\![P]\!]_{\mathcal{A}_1}$,

1. $Safe(H, A, C)$
2. For all K, B: $H, A, C \rightsquigarrow K, B, normal \implies K, B \in [\![Q]\!]_{\mathcal{A}_1}$

Provability $\vdash_{SL_1}$

$\{P\}C\{Q\}$ is SL_1-*provable* IFF: it can be derived using the inference rules of Separation logic and $\mathcal{A}_1$-valid assertions.

Soundness

$\vdash_{SL_1} \{P\}C\{Q\} \implies \models_{SL_1} \{P\}C\{Q\}$

Solving the Isolation Problem

$\mathcal{M}(H, A, (C_1, id_1), \dots, (C_n, id_n)) = H, A_{mash}, D_1; \dots; D_n$

Case A: Any location-property pair that is read during the reduction of D_i on H, A_{mash} is not accessed during the reduction of D_j on H, A_{mash}

Procedure 1 (sufficient for case A)

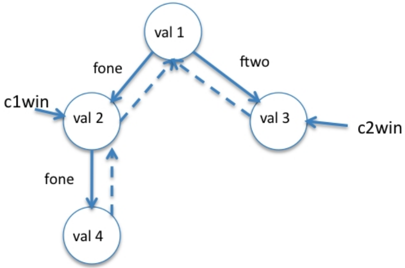
1. Deduce specifications $\{P_1\}C_1\{Q_1\}, \dots, \{P_n\}C_n\{Q_n\}$ in SL_1.
2. Show that $H, A \in [\![P_1 * \dots * P_n * true]\!]_{\mathcal{A}_1}$ holds.

Theorem

Procedure 1 is sound.

- Spent most of my time proving the above theorem.

Example: Tree access



$$H \quad := \quad \left\{ \begin{array}{lcl} l & : & \{val : 1, fone : l_1, ftwo : l_2\} \\ l_1 & : & \{val : 2, fone : l_3, par : l\} \\ l_2 & : & \{val : 3, fone : l_4, par : l\} \\ l_3 & : & \{val : 4, par : l_1\} \end{array} \right\}$$

Example: Tree access

$$\begin{array}{lcl} A_{mash} & := & \{c1win : l_1, c1tmp1 : 0, c1tmp2 : 0, c2win : l_2, c2tmp1 : 0, c2tmp2 : 0\} \\ C_1 & := & c1tmp1 = c1win.par; c1tmp2 = c1tmp1.val; c1win.val = c1tmp2+; \\ C_2 & := & c2tmp1 = c2win.val; c2win.val = c2tmp1 + 1; \end{array}$$

Prove Isolation.

Solution

1. $\{\exists x.c1win.val \mapsto 2 * c1win.par \mapsto x * x.val \mapsto 1\}C_1\{true\}$ $\{c2win.val \mapsto 3\}C_2\{true\}$
2. $H, A_{mash} \in [\![(\exists x.c1win.val \mapsto 2 * c1win.par \mapsto x * x.val \mapsto 1) * c2win.val \mapsto 3 * true]\!]_{\mathcal{A}_1}$

Issues

- Not in Algorithmic form
- Validity of $\mathcal{A}_1$ assertions is not decidable in general.
  - Yang and Calcagno (FSTTCS 2001 and a few others) have found decidable subsets of the assertion language.
  - I will explore these and their implications on procedure 1 in future.
- Nevertheless, we can carry out hand-proofs of isolation.

Outline

1. Background
   - Hoare Logic
   - Intuition behind Separation Logic
2. The Mashup Isolation problem ?
   - Formal Definition of Mashups
   - Isolation Property
   - How does Separation Logic help ?
3. Basic Separation Logic
   - Assertion language and Inference rules
   - Solving the Isolation Problem
4. Separation Logic with Permissions
   - Assertion language and Inference rules
   - Solving the Isolation Problem
5. Ongoing and Future Work

Separation Logic with Permissions SL_2

Assertion language $\mathcal{A}_2$

$$P := B \mid emp \mid x.p \overset{a}{\mapsto} E \mid P * P \mid P \mathrel{-\!\!*} P \mid P \implies P \mid \exists x.P$$

where $a \subseteq \{r, w\}$.

Satisfaction of Assertions

- Assertions are on heap-stores and also on actions performed by programs.
- Define permission maps Σ as $Loc \rightharpoonup \mathbb{P} \rightharpoonup \{\{r\}, \{w\}, \{r, w\}\}$
- General form: $H, A, \Sigma \models_{\mathcal{A}_2} P$

Satisfaction of Assertions ($\mathcal{A}_2$)

- Points-to: $H, A, \Sigma \models_{\mathcal{A}_2} E_1.p \overset{a}{\mapsto} E_2 \quad iff$
  - $dom(H) = \{([\![E_1]\!]_{Exp}A, p)\} = dom(\Sigma)$
  - $H([\![E_1]\!]_{Exp}A).p = [\![E_2]\!]_{Exp}A$
  - $a = \Sigma([\![E_1]\!]_{Exp}A).p$
- Separating-Conjunction:
  $H, A, \Sigma \models_{\mathcal{A}_2} P_1 * P_2 \quad iff \quad \exists H_1, \Sigma_1, H_2, \Sigma_2 :$
  - $H_1, A, \Sigma_1 \models_{\mathcal{A}_2} P_1$
  - $H_2, A, \Sigma_2 \models_{\mathcal{A}_2} P_2$
  - $H_1, \Sigma_1 \bowtie H_2, \Sigma_2$
  - $H_1 \cup H_2 = H \bigwedge \Sigma_1 \cup \Sigma_2 = \Sigma$

  where $dom(H_1), \Sigma_1 \bowtie dom(H_2), \Sigma_2$ means that
  - $H_1 \cup H_2, \Sigma_1 \cup \Sigma_2$ are defined
  - $\forall l, p \in dom(\Sigma_1) \cap dom(\Sigma_2) : \Sigma_1(l) = \Sigma_2(l) = \{r\}$

Observe that $(x.p \overset{r}{\mapsto} E) * (x.p \overset{r}{\mapsto} E) \Leftrightarrow x.p \overset{r}{\mapsto} E$.

Satisfaction of Assertions ($\mathcal{A}_2$)

- Points-to: $H, A, \Sigma \models_{\mathcal{A}_2} E_1.p \stackrel{a}{\mapsto} E_2 \quad iff$
  - $dom(H) = \{([\![E_1]\!]_{Exp}A, p)\} = dom(\Sigma)$
  - $H([\![E_1]\!]_{Exp}A).p = [\![E_2]\!]_{Exp}A$
  - $a = \Sigma([\![E_1]\!]_{Exp}A).p$
- Separating-Conjunction:
  $H, A, \Sigma \models_{\mathcal{A}_2} P_1 * P_2 \quad iff \quad \exists H_1, \Sigma_1, H_2, \Sigma_2 :$
  - $H_1, A, \Sigma_1 \models_{\mathcal{A}_2} P_1$
  - $H_2, A, \Sigma_2 \models_{\mathcal{A}_2} P_2$
  - $H_1, \Sigma_1 \bowtie H_2, \Sigma_2$
  - $H_1 \cup H_2 = H \bigwedge \Sigma_1 \cup \Sigma_2 = \Sigma$

  where $dom(H_1), \Sigma_1 \bowtie dom(H_2), \Sigma_2$ means that
  - $H_1 \cup H_2, \Sigma_1 \cup \Sigma_2$ are defined
  - $\forall l, p \in dom(\Sigma_1) \cap dom(\Sigma_2) : \Sigma_1(l) = \Sigma_2(l) = \{r\}$

Observe that $(x.p \stackrel{r}{\mapsto} E) * (x.p \stackrel{r}{\mapsto} E) \Leftrightarrow x.p \stackrel{r}{\mapsto} E$.

Satisfaction of Assertions ($\mathcal{A}_2$)

- Points-to: $H, A, \Sigma \models_{\mathcal{A}_2} E_1.p \overset{a}{\mapsto} E_2 \quad iff$
  - $dom(H) = \{([\![E_1]\!]_{Exp}A, p)\} = dom(\Sigma)$
  - $H([\![E_1]\!]_{Exp}A).p = [\![E_2]\!]_{Exp}A$
  - $a = \Sigma([\![E_1]\!]_{Exp}A).p$
- Separating-Conjunction:
  $H, A, \Sigma \models_{\mathcal{A}_2} P_1 * P_2 \quad iff \quad \exists H_1, \Sigma_1, H_2, \Sigma_2 :$
  - $H_1, A, \Sigma_1 \models_{\mathcal{A}_2} P_1$
  - $H_2, A, \Sigma_2 \models_{\mathcal{A}_2} P_2$
  - $H_1, \Sigma_1 \bowtie H_2, \Sigma_2$
  - $H_1 \cup H_2 = H \bigwedge \Sigma_1 \cup \Sigma_2 = \Sigma$

  where $dom(H_1), \Sigma_1 \bowtie dom(H_2), \Sigma_2$ means that
  - $H_1 \cup H_2, \Sigma_1 \cup \Sigma_2$ are defined
  - $\forall l, p \in dom(\Sigma_1) \cap dom(\Sigma_2) : \Sigma_1(l) = \Sigma_2(l) = \{r\}$

Observe that $(x.p \overset{r}{\mapsto} E) * (x.p \overset{r}{\mapsto} E) \Leftrightarrow x.p \overset{r}{\mapsto} E$.

Axioms and Inference rules (SL_2)

$$\frac{y \textit{ not free in } E}{\{present(x) \wedge \exists y : E.p \overset{r}{\mapsto} y\} x = E.p \{\exists y : E[y/x].p \overset{r}{\mapsto} x\}} \quad []$$

$$\{E_1.p \overset{w}{\mapsto} \_ \wedge Wt(E_2)\} E_1.p = E_2 \{E_1.p \overset{w}{\mapsto} E_2 \wedge Wt(E_2)\} \quad []$$

$$\frac{y \textit{ not free in any } E_i}{\{present(x) \wedge Wt(\tilde{E_i}) \wedge emp\} x := \{\tilde{p}_i : \tilde{E}_i\} \{\exists y : (x : \{p_i \overset{r,w}{\mapsto} E_i[y/x]\})\}} \quad []$$

- There are “backwards” version for each these using $-\!\!\ast$.
- All other rules stay the same.

Soundness

$$[\![P]\!]_{\mathcal{A}_2} \stackrel{def}{=} \{H, A \mid \exists\Sigma : H, A, \Sigma \models_{\mathcal{A}_2} P\}$$

Validity $\models_{SL_1}$

$\{P\}C\{Q\}$ is *SL_1-valid* IFF: for all H, A, Σ, $H, A, \Sigma \models_{\mathcal{A}_2} P$,

1. $Safe(H, A, C)$
2. For all K, B, D such that $H, A, C \rightsquigarrow K, B, D$,
   a. $K, B \in [\![Q]\!]_{\mathcal{A}_2}$ if $D = normal$.
   b. For all (l, p, a, v) IF $(l, p, a, v) \in Acc^{heap}((H, A, C), (K, B, D))$ and $l, p \in dom(H)$ THEN $a \in \Sigma(l).p$

Provability $\vdash_{SL_2}$

$\{P\}C\{Q\}$ is *SL_2-provable* IFF: it can be derived using the inference rules of Separation logic and $\mathcal{A}_2$-valid assertions.

Soundness: $\vdash_{SL_2} \{P\}C\{Q\} \implies \models_{SL_2} \{P\}C\{Q\}$.

Solving the Isolation Problem

Case B: Any location-property pair that is read during the reduction of D_i on H, A_{mash} is at most read during the reduction of on D_j on H, A_{mash}

Procedure 2 (sufficient for case B)

1. Deduce specifications $\{P_1\}C_1\{Q_1\}, \ldots, \{P_n\}C_n\{Q_n\}$ in SL_2.
2. Show that $H, A \in [\![P_1 * \ldots * P_n * true]\!]_{\mathcal{A}_2}$ holds.

Theorem

Procedure 2 is sound.

Handling Case C

Case C: Any location-property pair that is read during the reduction of D_i on H, A_{mash} is at most written and restored during the reduction of on D_j on H, A_{mash}

- Difficult to handle using permissions, we have an invariant which is temporarily broken and then restored.

Separation Logic with Information Hiding: POPL 2004

- **Main Idea**: Consider a program using multiple procedures.
  - Each procedure will have certain internal resources only managed by it.
  - The program should be reasoned about independent of these internal resources.
- Introduced the "hypothetical frame rule"

$$\frac{\begin{array}{c}\vdash \{P_1 * R\}C_1\{Q_1 * R\} \\ \cdots \\ \vdash \{P_n * R\}C_n\{Q_n * R\} \\ \{P_1\}k\{Q_1\}[X_1], \ldots, \{P_n\}k\{Q_n\} \vdash \{P\}C\{Q\}\end{array}}{\vdash \{P * R\}let\ k_1 = C_1, \ldots, k_n = C_n\ in\ C\{Q * R\}}$$

Handling Case C

Case C: Any location-property pair that is read during the reduction of D_i on H, A_{mash} is at most written and restored during the reduction of on D_j on H, A_{mash}

- Difficult to handle using permissions, we have an invariant which is temporarily broken and then restored.

Separation Logic with Information Hiding: POPL 2004

- **Main Idea**: Consider a program using multiple procedures.
  - Each procedure will have certain internal resources only managed by it.
  - The program should be reasoned about independent of these internal resources.
- Introduced the "hypothetical frame rule"

$$\frac{\begin{array}{c}\vdash \{P_1 * R\}C_1\{Q_1 * R\} \\ \ldots \\ \vdash \{P_n * R\}C_n\{Q_n * R\} \\ \{P_1\}k\{Q_1\}[X_1], \ldots, \{P_n\}k\{Q_n\} \vdash \{P\}C\{Q\}\end{array}}{\vdash \{P * R\} let\ k_1 = C_1, \ldots, k_n = C_n\ in\ C\{Q * R\}}[]$$

Our Procedure

- Our procedure is inspired by the hypothetical frame rule.
- Exact Assertion: R is exact iff for all stores S there is *unique* heap H such that $H, A \in [\![R]\!]_{\mathcal{A}_1}$.

Procedure 3 (sufficient for case C)

1. Deduce specifications of the form $\{P_1 * R\}C_1\{Q_1 * R\}, \ldots, \{P_n * R\}C_n\{Q_n * R\}$ in SL_1.
2. Show that R is exact.
3. Show that $H, A \in [\![P_1 * \ldots * P_n * R * true]\!]_{\mathcal{A}_1}$ holds.

Theorem

Procedure 3 is sound.

- Soundness of hypothetical frame rule used in the proof.
- *Ownership transfer* of R from C_1 *to* $\ldots$ *to* C_n.

Comparison with Authority Safety

Oakland 2010 paper:

- $Auth(H, A, C)$: Over-approximations of the set of actions performed during the reduction of C on H, A
- $AuthIsolation(H, A, C_1, \ldots, C_n)$: For all i, j, authority map of C_i does not contain a write action to a location where C_j reads from.
- Theorem: $AuthIsolation(H, A, C_1, \ldots, C_n) \implies Isolation(\mathcal{M}(H, A, C_1, \ldots, C_n))$

How does this relate to what we have done ?

- Semantically relates to solving Case B.
- Authority maps are usually computed by heap reachability analysis.
- Specifications are "more informative authority maps": More closely related to what is actually reached.

Comparison with Authority Safety

Oakland 2010 paper:

- $Auth(H, A, C)$: Over-approximations of the set of actions performed during the reduction of C on H, A
- $AuthIsolation(H, A, C_1, \ldots, C_n)$: For all i, j, authority map of C_i does not contain a write action to a location where C_j reads from.
- Theorem:
  $AuthIsolation(H, A, C_1, \ldots, C_n) \implies Isolation(\mathcal{M}(H, A, C_1, \ldots, C_n))$

How does this relate to what we have done ?

- Semantically relates to solving Case B.
- Authority maps are usually computed by heap reachability analysis.
- Specifications are "more informative authority maps": More closely related to what is actually reached.

Future Work: Migrating to *JavaScript*

Hopeful about the following features:

1. Deleting record properties: There are rules for *dispose*
2. Computable Properties x[E]: There are rules for handling pointer arithmetic (Reynolds, 2002).

Following seem tricky:

1. Dynamic addition of properties: Tempting to think of them as "resource allocation", but they are subtly different.
   - This is allocation of a particular resource and not a non-deterministically chosen one.
   - Frame rule might break !
   - May be some kind of "existence permissions" can help.
2. Prototype chains: How do we express what part of the chain is reached during property access ?

Future Work: Migrating to *JavaScript*

Hopeful about the following features:

1. Deleting record properties: There are rules for *dispose*
2. Computable Properties x[E]: There are rules for handling pointer arithmetic (Reynolds, 2002).

Following seem tricky:

1. Dynamic addition of properties: Tempting to think of them as "resource allocation", but they are subtly different.
   - This is allocation of a particular resource and not a non-deterministically chosen one.
   - Frame rule might break !
   - May be some kind of "existence permissions" can help.
2. Prototype chains: How do we express what part of the chain is reached during property access ?

Other Future directions

1. Think of better solutions for handling case C.
    - Explore if a permission based approach exists.
2. Formalize the notion of defensive consistency using separation logic.
    - Informally, defensively consistent functions are ones that are incorruptible by their clients.
    - Hypothetical frame rule can be useful.
3. Explore capability systems where meaning of a capability is specified using a specification rather than an authority map.
4. Formalize the right isolation property for mashups where each component is allowed to call methods defined by other components. Example: Yelp

Thank You