A Sip of the Chalice

Azalea Raad
Sophia Drossopoulou

Imperial College London
FTfJP 2011

26th July 2011

Chalice

- Verification tool for multi-threaded object-oriented style programs
  - Functional Correctness
  - Deadlock Prevention
- Built on Implicit Dynamic Frames
- Concurrency achieved through use of permissions
- No formal statement of syntax or semantics for assertions
- No soundness proof provided

Our Contributions

- Focus on a subset of Chalice: Chalice^f
- Concerned with functional correctness and not deadlock prevention mechanism
- Syntax and semantics of Chalice^f
- Distinguish "real" operations from "ghost" ones
- Parametric assertion language
- Verification conditions of Chalice^f through Hoare logic
- Soundness Proof

Introduction to Chalice: Permissions and Permission transfer

- A thread can access a location only if it has sufficient permissions
- Employ Boyland's fractional permissions
  - acc(x.f, n) where $0 \leq n \leq 1$

Introduction to Chalice: Permissions and Permission transfer

- A thread can access a location only if it has sufficient permissions
- Employ Boyland's fractional permissions
  - acc(x.f, n) where $0 \leq n \leq 1$
- Method precondition states required permissions

```
class PosInt{
   var i:Int
   invariant acc(this.i, 0.1) ∗ this.i>0

   method replace(y:Int)
      requires acc(this.i, 1) ∗ acc(y.i, 0.1);
      ensures acc(this.i, 1) ∗ acc(y.i, 0.1);
         {this.i := y.i;}
}
```

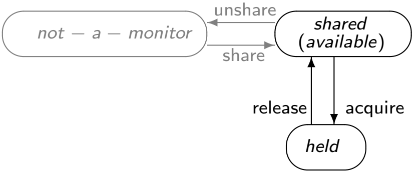
Introduction to Chalice: Permissions and Permission transfer

- A thread can access a location only if it has sufficient permissions
- Employ Boyland's fractional permissions
  - acc(x.f, n) where $0 \leq n \leq 1$
- Method precondition states required permissions

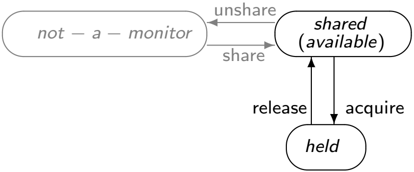
```
class PosInt{
  var i:Int
  invariant acc(this.i, 0.1) * this.i>0

  method replace(y:Int)
    requires acc(this.i, 1) * acc(y.i, 0.1);
    ensures acc(this.i, 1) * acc(y.i, 0.1);
      {this.i := y.i;}
}
```

- Precondition permissions transferred to callee upon method call
- Postcondition permissions returned to caller upon return

Introduction to Chalice: Monitors

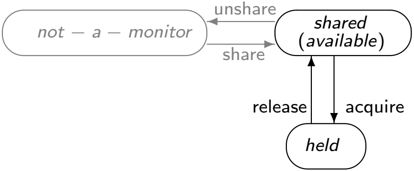


Introduction to Chalice: Monitors



- Each class associated with an *invariant* (assertion)
- Can be used as a mutual exclusion lock to synchronise access

Introduction to Chalice: Monitors



- Each class associated with an *invariant* (assertion)
- Can be used as a mutual exclusion lock to synchronise access
- Can be obtained and relinquished through `acquire` and `release`

```
class PosInt{
  var i:Int
  invariant acc(this.i, 0.1) * this.i>0
  void replace(y:Int) requires acc(this.i, 1) ...//as before
  void acquireAndReplace(y:Int)
    requires acc(this.i, 0.9) * acc(y.i, 0.1);
    ensures acc(this.i, 0.9) * acc(y.i, 0.1){
      {acquire this; this.replace(y); release this}
}
```

Introduction to Chalice: Threads

- New threads created using the `fork` statement
  τ_1: `fork tk:= x.m(ȳ);` //spawns new thread τ_2
- `tk`: A token used to identify the forked thread
- Permissions of `m`'s precondition transferred from τ_1 to τ_2

Introduction to Chalice: Threads

- New threads created using the `fork` statement
  τ_1: `fork tk:= x.m(ȳ);` //spawns new thread τ_2
- `tk`: A token used to identify the forked thread
- Permissions of `m`'s precondition transferred from τ_1 to τ_2
- Threads terminated through `join` statement and tokens
  τ_1: `join tk;` //terminates τ_2
- Permissions of `m`'s postcondition transferred from τ_2 to τ_1

Chalice^f Syntax

```
t::= CId                prog ::= class‾
class ::= CId → A × (FId → t) × (MId → meth)
meth ::= void m (t x‾) ( requires A ensures A ) {e}
e::= e;e |new CId() |x.f:=y |x:=y.f |x.m(y‾)
    |if(b) {e} else {e} | acquire x | release x
    | TkId := fork x.m(y‾) | join TkId
```

- Standard OO expressions
- Each class associated with an invariant (`A`)
- Method signature contains pre- and post-condition
- Thread creation (termination) through `fork` (`join`)
- Object monitor obtained (relinquished) through `acquire` (`release`)

Runtime Environment

$\texttt{P::= (e, ProcId, } \sigma \texttt{) | P|P}$

$\texttt{H : ObjAddr} \rightarrow \texttt{obj} \cup \texttt{TkAddr} \rightarrow \texttt{tk}$

$\texttt{obj : CId} \times \texttt{(FId} \rightarrow \texttt{value)} \times \texttt{ProcId}$

$\texttt{tk : } \{\texttt{TK}\} \times \texttt{(FId} \rightarrow \texttt{value)} \times \texttt{ProcId}$

$\Pi\texttt{: ProcId} \rightarrow \texttt{pMask}$

$\texttt{pMask} : \texttt{ObjAddr} \times \texttt{FId} \rightarrow \texttt{n} \cup \texttt{TkAddr} \times \texttt{FId} \rightarrow \texttt{n}$

$(\texttt{n} \in \texttt{Q} \wedge 0 \leq \texttt{n} \leq 1)$

$\sigma\texttt{::= (Var} \rightarrow \texttt{value)} \cup \texttt{(TkId} \rightarrow \texttt{TkAddr)}$

$\texttt{value::= null | ObjAddr | CId | MId | } \overline{\texttt{ObjAddr}}$

- Stack frame defined per process
- Monitor holder recorded for each object
- Token information stored in heap

  $\tau_1 : \ \texttt{tk} := \texttt{fork} \ \ \texttt{x.foo}(\bar{\texttt{y}}) \qquad \texttt{H(tk)} = (\texttt{TK}, \ \{\texttt{c} : \texttt{x.class}, \ \texttt{m} : \texttt{foo}, \ \texttt{args} : \texttt{x}.\bar{\texttt{y}}\}, \ \tau_2)$
- Global permission mask (Π) assigns a permission mask (π) to each process
- Permission mask (π) describes permission privileges held by thread

Assertion Language

$$\texttt{A::= acc(x.f, n) | x.f=v | } * \texttt{ | } \wedge \texttt{ | } \neg \texttt{ | ...}$$

- Support `acc(x.f, n), x.f=v`, $*, \wedge, \neg$
- Parametric with respect to connectives and their validity, subject to certain properties
- Agnostic to further connectives
- Assume existence of a validity judgement: $\mathsf{H}, \pi, \sigma \models \mathsf{A}$
- Assume existence of permission extraction function: $\mathcal{P}(\mathsf{H}, \sigma, \mathsf{A})$
- Assume presence of inference system $\mathtt{A}_1 \rightarrow_{\mathtt{a}} \mathtt{A}_2$

Assertion Language (Contd.)

Properties of $\models$ Judgement

- **R1.** $\mathsf{H}, \pi, \sigma \models \mathsf{A} \;\wedge\; \bar{\mathsf{y}} = \mathsf{fv}(\mathsf{A}) \;\wedge\; \sigma'(\bar{\mathsf{x}}) = \sigma(\bar{\mathsf{y}})$
  $\Longrightarrow\; \mathsf{H}, \pi, \sigma' \models \mathsf{A}[\bar{\mathsf{x}}/\bar{\mathsf{y}}]$

Assertion Language (Contd.)

Properties of $\models$ Judgement

- **R1.** $\mathsf{H}, \pi, \sigma \models \mathsf{A} \;\wedge\; \bar{\mathsf{y}} = \mathsf{fv}(\mathsf{A}) \;\wedge\; \sigma'(\bar{\mathsf{x}}) = \sigma(\bar{\mathsf{y}})$
  $\Longrightarrow\; \mathsf{H}, \pi, \sigma' \models \mathsf{A}[\bar{\mathsf{x}}/\bar{\mathsf{y}}]$
- **R2.** $\mathsf{H}, \pi, \sigma \models \mathsf{x.f} = \mathsf{y} \iff \mathsf{H}(\sigma(\mathsf{x}), \mathsf{f}) = \sigma(\mathsf{y})$

Assertion Language (Contd.)

Properties of $\models$ Judgement

- **R1.** $H, \pi, \sigma \models A \ \wedge \ \bar{y} = fv(A) \ \wedge \ \sigma'(\bar{x}) = \sigma(\bar{y})$
  $\implies H, \pi, \sigma' \models A[\bar{x}/\bar{y}]$
- **R2.** $H, \pi, \sigma \models x.f = y \iff H(\sigma(x), f) = \sigma(y)$
- **R3.** $H, \pi, \sigma \models acc(x.f, n) \iff \pi(\sigma(x), f) \geq n$

Assertion Language (Contd.)

Properties of $\models$ Judgement

- **R1.** $H, \pi, \sigma \models A \;\wedge\; \bar{y} = fv(A) \;\wedge\; \sigma'(\bar{x}) = \sigma(\bar{y})$
  $\implies H, \pi, \sigma' \models A[\bar{x}/\bar{y}]$
- **R2.** $H, \pi, \sigma \models x.f = y \iff H(\sigma(x), f) = \sigma(y)$
- **R3.** $H, \pi, \sigma \models acc(x.f, n) \iff \pi(\sigma(x), f) \geq n$
- **R4.** $H, \pi, \sigma \models A \wedge A' \iff H, \pi, \sigma \models A$ and $H, \pi, \sigma \models A'$

Assertion Language (Contd.)

Properties of $\models$ Judgement

- **R1.** $H, \pi, \sigma \models A \ \wedge \ \bar{y} = fv(A) \ \wedge \ \sigma'(\bar{x}) = \sigma(\bar{y})$
  $\implies H, \pi, \sigma' \models A[\bar{x}/\bar{y}]$
- **R2.** $H, \pi, \sigma \models x.f = y \iff H(\sigma(x), f) = \sigma(y)$
- **R3.** $H, \pi, \sigma \models acc(x.f, n) \iff \pi(\sigma(x), f) \geq n$
- **R4.** $H, \pi, \sigma \models A \wedge A' \iff H, \pi, \sigma \models A$ and $H, \pi, \sigma \models A'$
- **R5.** $H, \pi, \sigma \models A * A' \implies$
  $\forall(\iota.f)[\mathcal{P}(H, \sigma, A)(\iota.f) + \mathcal{P}(H, \sigma, A')(\iota.f) \leq 1]$
  $\wedge \ \forall(\kappa.g)[\mathcal{P}(H, \sigma, A)(\kappa.g) + \mathcal{P}(H, \sigma, A')(\kappa.g) \leq 1]$

Assertion Language (Contd.)

Properties of $\models$ Judgement

- **R1.** $H, \pi, \sigma \models A \;\wedge\; \bar{y} = fv(A) \;\wedge\; \sigma'(\bar{x}) = \sigma(\bar{y})$
  $\implies H, \pi, \sigma' \models A[\bar{x}/\bar{y}]$
- **R2.** $H, \pi, \sigma \models x.f = y \iff H(\sigma(x), f) = \sigma(y)$
- **R3.** $H, \pi, \sigma \models acc(x.f, n) \iff \pi(\sigma(x), f) \geq n$
- **R4.** $H, \pi, \sigma \models A \wedge A' \iff H, \pi, \sigma \models A$ and $H, \pi, \sigma \models A'$
- **R5.** $H, \pi, \sigma \models A * A' \implies$
  $\forall(\iota.f)[\mathcal{P}(H, \sigma, A)(\iota.f) + \mathcal{P}(H, \sigma, A')(\iota.f) \leq 1]$
  $\wedge \forall(\kappa.g)[\mathcal{P}(H, \sigma, A)(\kappa.g) + \mathcal{P}(H, \sigma, A')(\kappa.g) \leq 1]$
- **R6.** $A \rightarrow_a A' \;\wedge\; H, \pi, \sigma \models A \implies H, \pi, \sigma \models A'$

Assertion Language (Contd.)

Properties of $\models$ Judgement

- **R1.** $H, \pi, \sigma \models A \;\wedge\; \bar{y} = fv(A) \;\wedge\; \sigma'(\bar{x}) = \sigma(\bar{y})$
  $\Longrightarrow\; H, \pi, \sigma' \models A[\bar{x}/\bar{y}]$
- **R2.** $H, \pi, \sigma \models x.f = y \;\iff\; H(\sigma(x), f) = \sigma(y)$
- **R3.** $H, \pi, \sigma \models acc(x.f, n) \;\iff\; \pi(\sigma(x), f) \geq n$
- **R4.** $H, \pi, \sigma \models A \wedge A' \;\iff\; H, \pi, \sigma \models A$ and $H, \pi, \sigma \models A'$
- **R5.** $H, \pi, \sigma \models A * A' \;\Longrightarrow$
  $\forall(\iota.f)[\mathcal{P}(H, \sigma, A)(\iota.f) + \mathcal{P}(H, \sigma, A')(\iota.f) \leq 1]$
  $\wedge\; \forall(\kappa.g)[\mathcal{P}(H, \sigma, A)(\kappa.g) + \mathcal{P}(H, \sigma, A')(\kappa.g) \leq 1]$
- **R6.** $A \rightarrow_a A' \;\wedge\; H, \pi, \sigma \models A \;\Longrightarrow\; H, \pi, \sigma \models A'$
- **R7.** $H, \pi, \sigma \models true$

Assertion Language (Contd.)

Properties of $\mathcal{P}$ Function

- **R8.** $\mathcal{P}(\mathsf{H}, \sigma, \mathsf{A}) = \mathcal{P}(\mathsf{H}, \sigma[\overline{\mathsf{y}} \mapsto \overline{\sigma(\mathsf{x})}], \mathsf{A}[\overline{\mathsf{y}}/\overline{\mathsf{x}}])$

Assertion Language (Contd.)

Properties of $\mathcal{P}$ Function

- **R8.** $\mathcal{P}(\mathsf{H}, \sigma, \mathsf{A}) = \mathcal{P}(\mathsf{H}, \sigma[\bar{\mathsf{y}} \mapsto \overline{\sigma(\mathsf{x})}], \mathsf{A}[\bar{\mathsf{y}}/\bar{\mathsf{x}}])$
- **R9.** $\mathsf{H}, \pi, \sigma \models \mathsf{A} \implies$
  $\mathsf{H}, \mathcal{P}(\mathsf{H}, \sigma, \mathsf{A}), \sigma \models \mathsf{A}$
  $\wedge \forall(\iota.\mathsf{f})[\pi(\iota.\mathsf{f}) \geq \mathcal{P}(\mathsf{H}, \sigma, \mathsf{A})(\iota.\mathsf{f})]$

Assertion Language (Contd.)

Properties of $\mathcal{P}$ Function

- **R8.** $\mathcal{P}(\mathsf{H}, \sigma, \mathsf{A}) = \mathcal{P}(\mathsf{H}, \sigma[\bar{\mathsf{y}} \mapsto \overline{\sigma(\mathsf{x})}], \mathsf{A}[\bar{\mathsf{y}}/\bar{\mathsf{x}}])$
- **R9.** $\mathsf{H}, \pi, \sigma \models \mathsf{A} \implies$
  $\mathsf{H}, \mathcal{P}(\mathsf{H}, \sigma, \mathsf{A}), \sigma \models \mathsf{A}$
  $\wedge\ \forall(\iota.\mathsf{f})[\pi(\iota.\mathsf{f}) \geq \mathcal{P}(\mathsf{H}, \sigma, \mathsf{A})(\iota.\mathsf{f})]$
- **R10.** $\forall \mathsf{H}, \sigma, \mathsf{A}, (\iota.\mathsf{f})[\mathcal{P}(\mathsf{H}, \sigma, \mathsf{A})(\iota.\mathsf{f}) \neq \mathsf{Udf} \implies \iota.\mathsf{f} \in \mathsf{Dom}(\mathsf{H})]$
  $\wedge\ \forall \mathsf{H}, \sigma, \mathsf{A}, (\kappa.\mathsf{g})[\mathcal{P}(\mathsf{H}, \sigma, \mathsf{A})(\kappa.\mathsf{g}) \neq \mathsf{Udf} \implies \kappa.\mathsf{g} \in \mathsf{Dom}(\mathsf{H})]$

Hoare Logic and Semantics of Chalice^f

- Formalised verification conditions of Chalice^f through Hoare Logic
  {} acquire x {A[x/this]}   {A[x/this]} release x {}

A ≡ Invariant(x)

Hoare Logic and Semantics of Chalice^f

- Formalised verification conditions of Chalice^f through Hoare Logic
  $\{\}$ acquire x $\{A[x/this]\}$ $\{A[x/this]\}$ release x $\{\}$
- Divided semantics of Chalice^f into two parts
  - Operational semantics: "Real" execution of the program
    $P, H \rightsquigarrow P', H'$

AcqO
$$\frac{H(x)\downarrow_3 = \tau_g \qquad H' = H[(x)\downarrow_3 \mapsto \tau]}{(\text{acquire } x, \tau, \sigma), H \rightsquigarrow (\text{null}, \tau, \sigma), H'}$$

RelO
$$\frac{H(x)\downarrow_3 = \tau \qquad H' = H[(x)\downarrow_3 \mapsto \tau_g]}{(\text{release } x, \tau, \sigma), H \rightsquigarrow (\text{null}, \tau, \sigma), H'}$$

$A \equiv \text{Invariant}(x)$

Hoare Logic and Semantics of Chalice^f

- Formalised verification conditions of Chalice^f through Hoare Logic

  $\{\}$ acquire x $\{A[x/this]\}$ $\qquad$ $\{A[x/this]\}$ release x $\{\}$

- Divided semantics of Chalice^f into two parts
  - Operational semantics: "Real" execution of the program

    $P, H \rightsquigarrow P', H'$

    AcqO
    $$\frac{H(x)\downarrow_3 = \tau_g \qquad H' = H[(x)\downarrow_3 \mapsto \tau]}{(\text{acquire } x, \tau, \sigma), H \rightsquigarrow (\text{null}, \tau, \sigma), H'}$$

    RelO
    $$\frac{H(x)\downarrow_3 = \tau \qquad H' = H[(x)\downarrow_3 \mapsto \tau_g]}{(\text{release } x, \tau, \sigma), H \rightsquigarrow (\text{null}, \tau, \sigma), H'}$$

  - Permission passing semantics: "Ghost" operations necessary for soundness argument

    $P, H, \Pi \rightsquigarrow H', \Pi'$

    AcqP
    $$\frac{\mathcal{P}(H, \sigma, A[x/this]) = ps \quad \Pi' = \Pi[\tau + = ps, \quad \tau_g - = ps]}{(\text{acquire } x, \tau, \sigma), H, \Pi \rightsquigarrow H', \Pi'}$$

    RelP
    $$\frac{\mathcal{P}(H, \sigma, A[x/this]) = ps \quad \Pi' = \Pi[\tau - = ps, \quad \tau_g + = ps]}{(\text{release } x, \tau, \sigma), H, \Pi \rightsquigarrow H', \Pi'}$$

$A \equiv \text{Invariant}(x)$

Rewriting Rules of Chalice^f

- $P, H \rightsquigarrow P', H' \;\wedge\; P, H, \Pi \rightsquigarrow H', \Pi' \implies P, H, \Pi \rightsquigarrow P', H', \Pi'$

AcqO

$$\frac{H(x)\downarrow_3 = \tau_g \qquad H' = H[(x)\downarrow_3 \mapsto \tau]}{(\text{acquire } x, \tau, \sigma), H \rightsquigarrow (\text{null}, \tau, \sigma), H'}$$

AcqP

$$\frac{\mathcal{P}(H, \sigma, A[x/\text{this}]) = ps \qquad \Pi' = \Pi[\tau+ = ps, \quad \tau_g- = ps]}{(\text{acquire } x, \tau, \sigma), H, \Pi \rightsquigarrow H', \Pi'}$$

Acq

$$\frac{H(x)\downarrow_3 = \tau_g \qquad \mathcal{P}(H, \sigma, A[x/\text{this}]) = ps \qquad H' = H[(x)\downarrow_3 \mapsto \tau] \qquad \Pi' = \Pi[\tau+ = ps, \quad \tau_g- = ps]}{(\text{acquire } x, \tau, \sigma), H, \Pi \rightsquigarrow (\text{null}, \tau, \sigma), H', \Pi'}$$

Soundness of Chalice^f

Self-framing Assertions and Program Verification

- An assertion is self-framing if it contains sufficient permissions to check its validity.
  acc(x.f, 0.5) $*$ x.f=7 ✓   x.f=7 ✗

Soundness of Chalice^f

Self-framing Assertions and Program Verification

- An assertion is self-framing if it contains sufficient permissions to check its validity.
  acc(x.f, 0.5) $*$ x.f=7 ✓   x.f=7 ✗
- Definition of self-framing assertions is parametric to assertion language itself

Soundness of Chalice^f

Self-framing Assertions and Program Verification

- An assertion is self-framing if it contains sufficient permissions to check its validity.
  acc(x.f, 0.5) $*$ x.f=7 ✓ x.f=7 ✗
- Definition of self-framing assertions is parametric to assertion language itself
- $\text{Ver(Prog)} \iff \forall\ \text{C} \in \text{classes(Prog)},\ \forall \text{m} \in \text{methods(Prog, C)}$
  $[\text{SF(A)} \ \wedge\ \{\text{P}\}\ \text{e}\ \{\text{Q}\} \ \wedge\ \text{SF(P)} \ \wedge\ \text{SF(Q)}]$

  where $\text{P} \equiv \text{Pre(m)}$ $\text{Q} \equiv \text{Post(m)}$ $\text{e} \equiv \text{mBody(m)}$ $\text{A} \equiv \text{Invariant(C)}$

Soundness of Chalice^f

Well-formed Configuration and Program Verification

- $\mathsf{WF}(\mathsf{H}, \Pi, (\overline{\mathsf{e}, \tau, \sigma}^{1\ldots\mathsf{n}})) \iff$

- $\mathsf{WF}(\mathsf{H}, \Pi, (\overline{\mathsf{e}, \tau, \sigma}^{1\ldots n})) \iff$
  - **a.** Every token in the system is associated with a thread where its execution satisfies the post condition described by the token.

$$\begin{array}{l} \forall\ \kappa.[\mathsf{H}(\kappa) = (\mathsf{TK}, \{\mathsf{c} : \mathsf{C}, \mathsf{m} : \mathsf{m}, \mathsf{args} : \iota.\overline{\iota}\}, \tau) \implies \\ \quad \exists \mathsf{j} \in \{1 \ldots \mathsf{n}\}, \exists \mathsf{P}.[\tau = \tau_\mathsf{j} \ \wedge \ \ \mathsf{H}, \Pi, \tau_\mathsf{j}, \sigma_\mathsf{j} \models \mathsf{P} \\ \quad \wedge\ \{\mathsf{P}\}\ \mathsf{e_j}\ \{\mathsf{Post(m)}\} \wedge \sigma_\mathsf{j}(\mathsf{this}) = \iota \wedge \sigma_\mathsf{j}(\overline{\mathsf{u}}) = \overline{\iota}\ ] \end{array}$$

Soundness of Chalice^f

Well-formed Configuration and Program Verification

- $\mathsf{WF}(H, \Pi, (\overline{e, \tau, \sigma}^{1\ldots n})) \iff$
  - **a.** Every token in the system is associated with a thread where its execution satisfies the post condition described by the token.

    $$\forall\ \kappa.[H(\kappa) = (TK, \{c : C, m : m, args : \iota.\overline{\iota}\}, \tau) \implies \exists j \in \{1\ldots n\}, \exists P.[\tau = \tau_j \ \wedge \ H, \Pi, \tau_j, \sigma_j \models P \wedge \{P\}\ e_j\ \{Post(m)\} \wedge \sigma_j(this) = \iota \wedge \sigma_j(\overline{u}) = \overline{\iota}\ ]$$

  - **b.** For each location in the heap, the sum of permissions held by threads in the system does not exceed 1.

    $$\forall \iota.f[\textstyle\sum_{\tau \in DOM(\Pi)} \Pi(\tau)(\iota.f) \ \leq 1] \quad \wedge \quad \forall \kappa.g[\textstyle\sum_{\tau \in DOM(\Pi)} \Pi(\tau)(\kappa.g) \ \leq 1]$$

Soundness of Chalice^f

Well-formed Configuration and Program Verification

- $\mathsf{WF}(\mathsf{H}, \Pi, (\overline{\mathsf{e}, \tau, \sigma}^{1\ldots n})) \iff$
  - **a.** Every token in the system is associated with a thread where its execution satisfies the post condition described by the token.

    $$\forall\ \kappa.[\mathsf{H}(\kappa) = (\mathsf{TK}, \{\mathsf{c} : \mathsf{C}, \mathsf{m} : \mathsf{m}, \mathsf{args} : \iota.\overline{\iota}\}, \tau) \implies \exists \mathsf{j} \in \{1 \ldots n\}, \exists \mathsf{P}.[\tau = \tau_{\mathsf{j}} \ \wedge \ \ \mathsf{H}, \Pi, \tau_{\mathsf{j}}, \sigma_{\mathsf{j}} \models \mathsf{P} \wedge \{\mathsf{P}\}\ \mathsf{e}_{\mathsf{j}}\ \{\mathsf{Post}(\mathsf{m})\} \wedge \sigma_{\mathsf{j}}(\mathsf{this}) = \iota \wedge \sigma_{\mathsf{j}}(\overline{\mathsf{u}}) = \overline{\iota}\ ]$$

  - **b.** For each location in the heap, the sum of permissions held by threads in the system does not exceed 1.

    $$\forall \iota.\mathsf{f}[\textstyle\sum_{\tau \in \mathsf{DOM}(\Pi)} \Pi(\tau)(\iota.\mathsf{f}) \ \leq 1] \quad \wedge \quad \forall \kappa.\mathsf{g}[\textstyle\sum_{\tau \in \mathsf{DOM}(\Pi)} \Pi(\tau)(\kappa.\mathsf{g}) \ \leq 1]$$

  - **c.** The invariants of *shared* objects hold and are pairwise separate.

    $$\mathsf{H}, \Pi, \overline{\mathsf{x} \mapsto \iota}^{1\ldots m}, \tau_{\mathsf{g}} \models \overline{*\mathsf{Invariant}(\iota_{\mathsf{i}}.\mathsf{class})[\mathsf{x}_{\mathsf{i}}/\mathsf{this}]}^{1\ldots m} \quad \text{for} \quad \overline{\iota}^{1\ldots m} = \{\iota | \iota \in \mathsf{Dom}(\mathsf{H}) \wedge \mathsf{H}(\iota)\downarrow_3 = \tau_{\mathsf{g}}\}$$

Soundness of Chalice^f

Well-formed Configuration and Program Verification

- $\mathsf{WF}(\mathsf{H}, \Pi, (\overline{\mathsf{e}, \tau, \sigma}^{1\ldots n})) \iff$
  - **a.** Every token in the system is associated with a thread where its execution satisfies the post condition described by the token.

    $$\begin{array}{l}\forall\ \kappa.[\mathsf{H}(\kappa) = (\mathsf{TK}, \{\mathsf{c} : \mathsf{C}, \mathsf{m} : \mathsf{m}, \mathsf{args} : \iota.\bar{\iota}\}, \tau) \implies \\ \quad \exists \mathsf{j} \in \{1 \ldots \mathsf{n}\}, \exists \mathsf{P}.[\tau = \tau_\mathsf{j} \ \wedge \ \ \mathsf{H}, \Pi, \tau_\mathsf{j}, \sigma_\mathsf{j} \models \mathsf{P} \\ \quad \wedge\ \{\mathsf{P}\}\ \mathsf{e_j}\ \{\mathsf{Post(m)}\} \wedge \sigma_\mathsf{j}(\mathsf{this}) = \iota \wedge \sigma_\mathsf{j}(\bar{\mathsf{u}}) = \bar{\iota}\ ]\end{array}$$

  - **b.** For each location in the heap, the sum of permissions held by threads in the system does not exceed 1.

    $$\forall \iota.\mathsf{f}[\textstyle\sum_{\tau \in \mathsf{DOM}(\Pi)} \Pi(\tau)(\iota.\mathsf{f}) \ \leq 1] \quad \wedge \quad \forall \kappa.\mathsf{g}[\textstyle\sum_{\tau \in \mathsf{DOM}(\Pi)} \Pi(\tau)(\kappa.\mathsf{g}) \ \leq 1]$$

  - **c.** The invariants of *shared* objects hold and are pairwise separate.

    $$\begin{array}{l}\mathsf{H}, \Pi, \overline{\mathsf{x} \mapsto \iota}^{1\ldots m}, \tau_\mathsf{g} \models \overline{*\mathsf{Invariant}(\iota_\mathsf{i}.\mathsf{class})[\mathsf{x_i}/\mathsf{this}]}^{1\ldots m} \\ \quad \mathsf{for} \quad \bar{\iota}^{1\ldots m} = \{\iota | \iota \in \mathsf{Dom(H)} \wedge \mathsf{H}(\iota) \downarrow_3 = \tau_\mathsf{g}\}\end{array}$$

  - **d.** Each thread is associated with at most one token.

    $$\forall \kappa, \kappa'[\mathsf{H}(\kappa) \downarrow_3 = \tau \ \wedge \ \mathsf{H}(\kappa') \downarrow_3 = \tau \implies \kappa = \kappa']$$

Soundness of Chalice^f (Contd.)

Soundness Theorem

- Theorem 1

$$
\begin{aligned}
&\mathsf{Ver(Prog)}\\
&\wedge\ \mathsf{WF}(\mathsf{H}, \Pi, (\overline{\mathsf{e}, \tau, \sigma}^{0\ldots n}))\\
&\wedge\ (\{\mathsf{P_i}\}\ \mathsf{e_i}\ \{\mathsf{Q_i}\})_{i\in\{0\ldots n\}}\\
&\wedge\ (\mathsf{H}, \Pi, \sigma_i, \tau_i \models \mathsf{P_i})_{i\in\{0\ldots n\}}\\
&\wedge\ (\mathsf{e_0}, \tau_0, \sigma_0)|(\overline{\mathsf{e}, \tau, \sigma}^{1\ldots n}), \mathsf{H}, \Pi \rightsquigarrow (\mathsf{e}'_0, \tau_0, \sigma'_0)|(\overline{\mathsf{e}, \tau, \sigma}^{1\ldots n}), \mathsf{H}', \Pi'\\
&\qquad\qquad\Longrightarrow\\
&1.\ (\mathsf{H}', \Pi', \sigma_i, \tau_i \models \mathsf{P_i})_{i\in\{1\ldots n\}}\\
&2.\ \exists \mathsf{P}'_0.[\ \{\mathsf{P}'_0\}\ \mathsf{e}'_0\ \{\mathsf{Q_0}\}\ \wedge\ \mathsf{H}', \Pi', \sigma'_0, \tau_0 \models \mathsf{P}'_0\ ]\\
&3.\ \mathsf{WF}(\mathsf{H}', \Pi', (\mathsf{e}'_0, \tau_0, \sigma'_0)|(\overline{\mathsf{e}, \tau, \sigma}^{1\ldots n})\ )
\end{aligned}
$$

Soundness of Chalice^f (Contd.)

Soundness Theorem

- Theorem 1

  Ver(Prog)

  $\wedge\ \mathsf{WF}(\mathsf{H}, \Pi, (\overline{\mathsf{e}, \tau, \sigma}^{0\ldots n}))$

  $\wedge\ (\{\mathsf{P_i}\}\ \mathsf{e_i}\ \{\mathsf{Q_i}\})_{i \in \{0 \ldots n\}}$

  $\wedge\ (\mathsf{H}, \Pi, \sigma_i, \tau_i \models \mathsf{P_i})_{i \in \{0 \ldots n\}}$

  $\wedge\ (\mathsf{e_0}, \tau_0, \sigma_0) | (\overline{\mathsf{e}, \tau, \sigma}^{1\ldots n}), \mathsf{H}, \Pi \rightsquigarrow (\mathsf{e}'_0, \tau_0, \sigma'_0) | (\overline{\mathsf{e}, \tau, \sigma}^{1\ldots n}), \mathsf{H}', \Pi'$

  $$\Longrightarrow$$

  1. $(\mathsf{H}', \Pi', \sigma_i, \tau_i \models \mathsf{P_i})_{i \in \{1 \ldots n\}}$
  2. $\exists \mathsf{P}'_0 . [\ \{\mathsf{P}'_0\}\ \mathsf{e}'_0\ \{\mathsf{Q_0}\}\ \wedge\ \mathsf{H}', \Pi', \sigma'_0, \tau_0 \models \mathsf{P}'_0\ ]$
  3. $\mathsf{WF}(\mathsf{H}', \Pi', (\mathsf{e}'_0, \tau_0, \sigma'_0) | (\overline{\mathsf{e}, \tau, \sigma}^{1\ldots n})\ )$

- Two more theorems covering thread creation and termination

Soundness of Chalice^f (Contd.)

Soundness Theorem

- Theorem 1

  Ver(Prog)
  $\wedge\ \mathsf{WF}(\mathsf{H}, \Pi, (\overline{\mathsf{e}, \tau, \sigma}^{0\ldots n}))$
  $\wedge\ (\{\mathsf{P_i}\}\ \mathsf{e_i}\ \{\mathsf{Q_i}\})_{i \in \{0\ldots n\}}$
  $\wedge\ (\mathsf{H}, \Pi, \sigma_i, \tau_i \models \mathsf{P_i})_{i \in \{0\ldots n\}}$
  $\wedge\ (\mathsf{e_0}, \tau_0, \sigma_0) | (\overline{\mathsf{e}, \tau, \sigma}^{1\ldots n}), \mathsf{H}, \Pi \rightsquigarrow (\mathsf{e}'_0, \tau_0, \sigma'_0) | (\overline{\mathsf{e}, \tau, \sigma}^{1\ldots n}), \mathsf{H}', \Pi'$
  $$\Longrightarrow$$
  1. $(\mathsf{H}', \Pi', \sigma_i, \tau_i \models \mathsf{P_i})_{i \in \{1\ldots n\}}$
  2. $\exists \mathsf{P}'_0 .[\ \{\mathsf{P}'_0\}\ \mathsf{e}'_0\ \{\mathsf{Q_0}\}\ \wedge\ \mathsf{H}', \Pi', \sigma'_0, \tau_0 \models \mathsf{P}'_0\ ]$
  3. $\mathsf{WF}(\mathsf{H}', \Pi', (\mathsf{e}'_0, \tau_0, \sigma'_0) | (\overline{\mathsf{e}, \tau, \sigma}^{1\ldots n})\ )$
- Two more theorems covering thread creation and termination
- Proof by induction supported by auxiliary lemmas

Conclusions

- A simplified and succinct sub-syntax of Chalice
- Parametric assertion language
- Formalised its Hoare logic and semantics, established its soundness
- Separation of "real" and "ghost" operations

Future Work

- Expand Chalice^f to incorporate deadlock prevention mechanism
- Implement Heule et al's approach of fractional permissions without fractions[1]

[1] Fractional Permissions without Fractions - S. Heule, R. Leino, P. Müller, A. J. Summers - FTfJP'11

Questions?

- `void questions() requires knowledge ensures answer;`

FAss $\{acc(x.f, 1)\}\ x.f := y\ \{acc(x.f, 1) * x.f = y\}$

If B is a case-split assert.

$$\frac{\{B \land P\}\ C1\ \{Q\} \quad \{\neg B \land P\}\ C2\ \{Q\}}{\{P\}\ if(B)\ then\ C1\ else\ C2\ \{Q\}}$$

VAss

$$\frac{z > 0}{\{acc(x.f, z)\}\ y := x.f\ \{acc(x.f, z) * y = x.f\}}$$

Meth

$$\frac{Pre(m) = P(\overline{u}) \qquad Post(m) = Q(\overline{w})}{\{P[x/this][\overline{y}/\overline{u}]\}\ \mathbf{x.m(\overline{y})}\ \{Q[x/this][\overline{y}/\overline{w}]\}}$$

Val.

$$\frac{SF(P)}{\{P\}\ v\ \{P\}}$$

New

$$\frac{f_i \in FS(C)}{\{\}\ x := new\ C\ \{*acc(x.f_i, 1) * x.f_i = null\}}$$

Seq. $$\frac{\{P\}\ C_1\ \{Q\} \qquad \{Q\}\ C_2\ \{R\}}{\{P\}\ C_1; C_2\ \{R\}}$$

Acq. $$\{\}\ \text{acquire } x\ \{A[x/this]\}$$

Fork $$\frac{Pre(m) = P(\bar{u}) \qquad x.class = C}{\{P[x/this][\bar{y}/\bar{u}]\}\ \textbf{fork tk} := \textbf{x.m}(\bar{\textbf{y}})\ \{Thread(tk, C, m, x.\bar{y})\}}$$

Rel. $$\{A[x/this]\}\ \text{release } x\ \{\}$$

Join $$\frac{Post(m) = A(\bar{u})}{\{Thread(tk, C, m, x.\bar{y})\}\ \text{join } tk\ \{Post(m)[x/this][\bar{y}/\bar{u}]\}}$$

Con. $$\frac{\begin{array}{lll} SF(P) & SF(Q) & \\ P \rightarrow_a P' & \{P'\}\ C\ \{Q'\} & Q' \rightarrow_a Q \end{array}}{\{P\}\ C\ \{Q\}}$$

Frm $$\frac{\{P\}\ C\ \{Q\} \quad SF(R)}{\{P * R\}\ C\ \{Q * R\}}\ FV(R) \cap Mods(C) = \emptyset$$

Operational Semantics of Chalice^f

FAssO
$$\frac{\sigma(x) = \iota \qquad H' = H[\iota \mapsto H(\iota)[f \mapsto \sigma(y)]]}{(x.f := y\ ,\ \tau, \sigma), H \leadsto (\sigma(y), \tau, \sigma), H'}$$

VAssO
$$\frac{\sigma(x) = \iota \qquad H(\iota) \downarrow_2 (f) = v \qquad \sigma' = \sigma[y \mapsto v]}{(y := x.f,\ \tau, \sigma), H \leadsto (v, \tau, \sigma'), H}$$

IfTO
$$\frac{}{(\text{if}(\text{true})\text{then}\{e2\}\text{else}\{e3\}, \tau, \sigma), H \leadsto (e2, \tau, \sigma), H}$$

IfFO
$$\frac{}{(\text{if}(\text{false})\text{then}\{e2\}\text{else}\{e3\}, \tau, \sigma), H \leadsto (e3, \tau, \sigma), H}$$

ValO
$$\frac{}{(v; e, \tau, \sigma), H \leadsto (e, \tau, \sigma), H}$$

MethO
$$\frac{tk \notin \sigma}{(x.m(\bar{y}), \tau, \sigma), H \leadsto ((\text{fork } tk := x.m(\bar{y}); \text{join } tk), \tau, \sigma), H}$$

Operational semantics of Chalice^f (Contd.)

SeqO
$$\frac{(e_1, \tau, \sigma), H \rightsquigarrow (e_1', \tau, \sigma'), H'}{(e_1; e_2, \tau, \sigma), H \rightsquigarrow (e_1'; e_2, \tau, \sigma'), H'}$$

NewO
$$\frac{\begin{array}{l} FS(C) = \{t_1\ f_1 \ldots, t_r\ f_r\} \quad \iota \notin H \quad \sigma' = \sigma[x \mapsto \iota] \\ H' = H[\iota \mapsto (C, \{f_1 : null, \ldots, f_r : null\},\ \tau)] \end{array}}{(x := new\ C\ , \tau, \sigma), H \rightsquigarrow (\iota, \tau, \sigma'), H'}$$

AcqO
$$\frac{\begin{array}{l} \sigma(x) = \iota \qquad H(\iota) \downarrow_3 = \tau_g \\ H' = H[\iota \mapsto (H(\iota) \downarrow_1, H(\iota) \downarrow_2, \tau)] \end{array}}{(acquire\ x, \tau, \sigma), H \rightsquigarrow (null, \tau, \sigma), H'}$$

RelO
$$\frac{\begin{array}{l} \sigma(x) = \iota \qquad H(\iota) \downarrow_3 = \tau \\ H' = H[\iota \mapsto (H(\iota) \downarrow_1, H(\iota) \downarrow_2, \tau_g)] \end{array}}{(release\ x, \tau, \sigma), H \rightsquigarrow (null, \tau, \sigma), H'}$$

Operational Semantics of Chalice^f (Contd.)

ForkO

$$\frac{\begin{array}{l}\kappa \notin dom(H) \quad \tau' \notin range(H) \quad mBody(m) = e(\overline{u}) \\ \sigma(x) = \iota \quad H(\iota)\downarrow_1 = C \quad \overline{\sigma(y)} = \overline{\iota} \\ H' = H[\kappa \mapsto (TK, \{c : C, m : m, args : \iota.\overline{\iota}\}, \tau')] \\ \sigma' = \sigma[tk \mapsto \kappa] \qquad \sigma'' = this \mapsto \iota, \overline{u} \mapsto \overline{\iota}\end{array}}{(fork\ tk := x.m(\overline{y}), \tau, \sigma), H \rightsquigarrow ((null, \tau, \sigma')|(e, \tau', \sigma'')), H'}$$

JoinO

$$\frac{H(\sigma(tk))\downarrow_3 = \tau' \qquad H' = H[\sigma(tk) \mapsto \epsilon]}{(v, \tau', \sigma')|(join\ tk, \tau, \sigma), H \rightsquigarrow (null, \tau, \sigma), H'}$$

ThrdO

$$\frac{\overline{P'}, H \rightsquigarrow \overline{P''}, H'}{\overline{P_1} \mid \overline{P'} \mid \overline{P_2}, H \rightsquigarrow \overline{P_1} \mid \overline{P''} \mid \overline{P_2}, H',}$$

Permission Passing Semantics of Chalice^f

RestP

$$\frac{\begin{array}{l} \mathsf{e} \in \{\, \mathtt{y := x.f}, \mathtt{x.f := y}, \mathtt{if}\ldots, \mathtt{x.m}(\overline{\mathtt{y}}) \\ (\mathsf{e}\ ,\ \tau, \sigma), \mathsf{H} \rightsquigarrow (\mathsf{e}', \tau, \sigma), \mathsf{H}' \end{array}}{(\mathsf{e}\ ,\ \tau, \sigma), \mathsf{H} \rightsquigarrow \mathsf{H}', \Pi}$$

NewP

$$\frac{\begin{array}{l} \mathsf{FS}(\mathsf{C}) = \{\mathsf{t}_1\ \mathsf{f}_1 \ldots, \mathsf{t}_r\ \mathsf{f}_r\} \qquad \iota = \mathsf{dom}(\mathsf{H}') \setminus \mathsf{dom}(\mathsf{H}) \\ \Pi' = \Pi[(\tau)(\iota, \mathsf{f}_i) \mapsto 1]_{i \in 1 \ldots r} \end{array}}{(\mathsf{x := new\ C}\ , \tau, \sigma), \mathsf{H}, \Pi \rightsquigarrow \mathsf{H}', \Pi'}$$

AcqP

$$\frac{\begin{array}{l} \sigma(\mathsf{x}) = \iota \qquad \mathsf{H}(\iota) \downarrow_1 = \mathsf{C} \\ \mathsf{Invariant}(\mathsf{C}) = \mathsf{A} \qquad \mathcal{P}(\mathsf{H}, \sigma, \mathsf{A}[\mathsf{x}/\mathsf{this}]) = \mathsf{ps} \\ \Pi' = \Pi[\tau + = \mathsf{ps}, \quad \tau_g - = \mathsf{ps}] \end{array}}{(\mathsf{acquire\ x}, \tau, \sigma), \mathsf{H}, \Pi \rightsquigarrow \mathsf{H}', \Pi'}$$

RelP

$$\frac{\begin{array}{l} \sigma(\mathsf{x}) = \iota \qquad \mathsf{H}(\iota) \downarrow_1 = \mathsf{C} \\ \mathsf{Invariant}(\mathsf{C}) = \mathsf{A} \qquad \mathcal{P}(\mathsf{H}, \sigma, \mathsf{A}[\mathsf{x}/\mathsf{this}]) = \mathsf{ps} \\ \Pi' = \Pi[\tau - = \mathsf{ps}, \quad \tau_g + = \mathsf{ps}] \end{array}}{(\mathsf{release\ x}, \tau, \sigma), \mathsf{H}, \Pi \rightsquigarrow \mathsf{H}', \Pi'}$$

Permission Semantics of Chalice^f (Contd.)

ForkP

$$\frac{\begin{array}{l}\sigma(x) = \iota \quad H(\iota)\downarrow_1 = C \quad pre(C, m) = A(\bar{u}) \\ \mathcal{P}(H, \sigma, A[x/this][\bar{x}/\bar{u}]) = ps \\ \kappa \in H' \setminus H \quad H(\kappa)\downarrow_3 = \tau' \\ \Pi'' = \Pi[\tau' \mapsto ps, \ \ \tau - = ps] \\ \Pi' = \Pi''[(\tau)(\kappa.g) \mapsto 1] \ \text{ for } g \in \{c, m, args\}\end{array}}{(\text{fork tk} := x.m(\bar{y}), \tau, \sigma), H, \Pi \rightsquigarrow H', \Pi'}$$

JoinP

$$\frac{\begin{array}{l}\sigma(tk) = \kappa \quad H(\kappa)\downarrow_3 = \tau' \quad H(\kappa.args) = \iota.\bar{\iota} \\ H(\kappa.c) = C \quad H(\kappa.m) = m \quad Post(C, m) = A(\bar{u}) \\ \mathcal{P}(H, [this \mapsto \iota, \bar{u} \mapsto \bar{\iota}], A(\bar{u})) = ps \\ \Pi' = \Pi[\tau + = ps, \ \ \tau' - = ps]\end{array}}{(v, \tau', \sigma) | (\text{join tk}, \tau, \sigma), H, \Pi \rightsquigarrow H', \Pi'}$$

ThrdP

$$\frac{P, H, \Pi \rightsquigarrow \overline{P'}, H', \Pi'}{\overline{P_1} \mid P \mid \overline{P_2}, H, \Pi \rightsquigarrow \overline{P_1} \mid \overline{P'} \mid \overline{P_2}, H', \Pi'}$$

Permission Passing Semantics of Chalice^f (Contd.)

SeqP $$\frac{(\mathsf{e}_1, \tau, \sigma), \mathsf{H}, \Pi \rightsquigarrow (\mathsf{e}_1', \tau, \sigma'), \mathsf{H}', \Pi'}{(\mathsf{e}_1; \mathsf{e}_2, \tau, \sigma), \mathsf{H}, \Pi \rightsquigarrow \mathsf{H}', \Pi'}$$