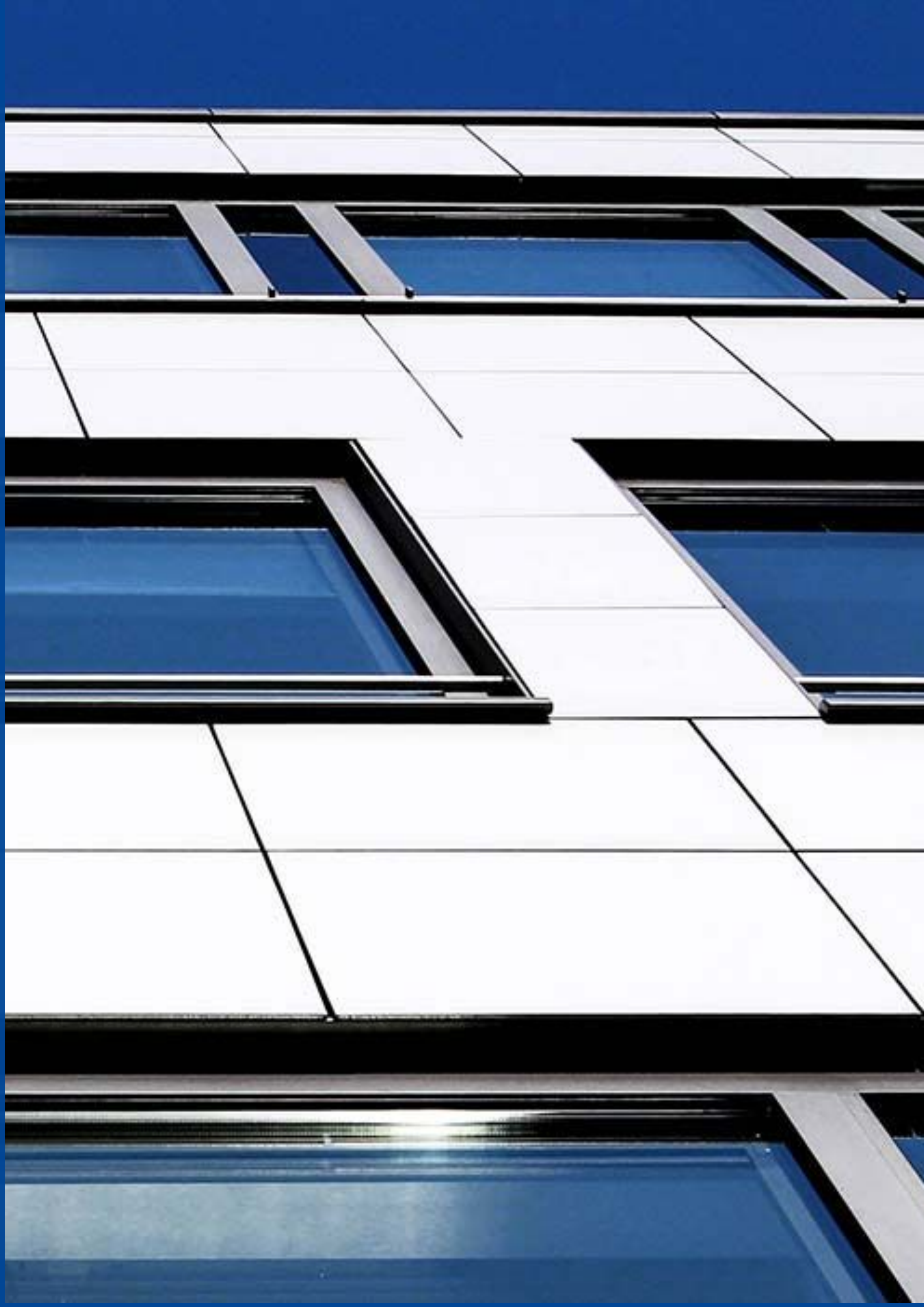
fortiss

A Generic Framework for Analyzing Java Programs

Chuangjie Xu (j.w.w. Ulirch Schöpp)

MSP 101 Seminar, Strathclyde, 20 May 2021



Enforcing Secure Programming Guidelines

- Does my Java program follow secure programming guidelines such as
  - *“All inputs must be sanitized.”*
  - *“Any access to sensitive data must be authorized.”*
  - *“Any access to sensitive data must be logged.”*
  - …
- Can guidelines be verified continuously and incrementally?
- A lightweight tool for this can help programmers to avoid making typical errors during the development.


```java
@Service
@Transactional
public class ParticipantService implements IParticipantService {

  /**
   * How long an unconfirmed registration is kept.
   */
  private static int EXPIRATION_TIME_HOURS = 48;

  /**
   * Number of accepted unconfirmed registrations with the same e-mail address.
   */
  private static int MOST_UNCONFIRMED_REGS = 2;

  /**
   * Threshold of the number of registrations per day that counts as unusual.
   */
  private static int UNUSUAL_THRESHOLD = 200;

  @Autowired
  private Environment env;

  @Autowired
  private ParticipantRepository participantRepository;

  private static final Logger log = LoggerFactory.getLogger(ParticipantService.class);

  @Override
  public Participant newRegistration(ParticipantDto participantDto) throws TooManyUnconfirmedRegs {
    String token = UUID.randomUUID().toString();
    int unconfirmed = participantRepository.countUnconfirmedByEmail(participantDto.getEmail());

    if (unconfirmed > MOST_UNCONFIRMED_REGS) {
      throw new TooManyUnconfirmedRegs(unconfirmed);
    }

    Participant p = new Participant(participantDto);
    p.setNeedsConfirmation(token);
    p.setRegistrationDate(new Date());

    return participantRepository.save(p);
  }

  @Override
  public void confirmRegistration(String verificationToken) {
    Participant participant = getParticipant(verificationToken);
    if (participant == null) {
      return;
    }
    Participant confirmed = participantRepository.findConfirmed(participant.getEmail());
    if (confirmed != null) {
      participantRepository.delete(confirmed);
    }

    participant.setConfirmed();
    participantRepository.save(participant);
  }

  @Override
  public void cancelRegistration(String verificationToken) {
    Participant participant = getParticipant(verificationToken);
    if (participant == null || participant.isConfirmed()) {
      return;
    }

    participantRepository.delete(participant);
  }

  @Override
  public Participant getParticipant(String verificationToken) {
    return participantRepository.findByToken(verificationToken);
  }

  @Override
  public Participant getConfirmedRegistration(String email) {
    return participantRepository.findConfirmed(email);
-:--- ParticipantService.java   12% (46,0)   (Java/l Abbrev)
Font set for java-mode-default face.
```

GuideForce[1]

GuideForce develops effect type systems for lightweight static analysis.

- Imagine that functions of interests emit events when they are executed.
  - E.g., `Server.login()` emits a *login* event; `Connection.close()` emits a *close* event; …
  - Each execution of a program generates a (finite or infinite) trace of events.
  - Guidelines (of safety and liveness properties) specify which event traces are allowed.
- The type system has effect annotations to give information about the possible traces.
  - E.g., `login() ? readData() : close();` : type & {*login read*, *login close*}
- Inferring the type of a program is to compute its effect.
- If the effect is “contained” in the guideline, then the program adheres to the guideline.

[1] GuideForce (DFG 250888164) was Initiated by Martin Hofmann at LMU, and is now hosted at fortiss.

Example

Guideline 1: Any *access* to sensitive data must be *authorized*.

Server.java

```java

  void serve() {
    while (hasQuery()) {
      boolean authorized = verifyAuthorization();  // emits auth
      if (authorized) {
        (this:EntryPoint, authorized:Base) & {auth, auth access}* · {auth}
      }
    }
    (this:EntryPoint, authorized:Base) & {auth, auth access}*
  }

```

- Analysis Results	
- serve	This method adheres to the given guideline. Terminating effect: $\{auth, auth\ access\}^* \cdot \{log\}$ Nonterminating effect: $\{auth, auth\ access\}^\omega$

Example

Guideline 2: Any *access* to sensitive data must be *logged*.

Server.java

```
void serve() {
  while (hasQuery()) {
    boolean authorized = verifyAuthorization(); // emits auth
    if (authorized) {
      readSensitiveData(); // emits access
    }
  }
  logAccess(); // emits log
}
```

Generated trace: $(auth\ access)^{\omega}$

- Analysis Results
  - serve

This method may not satisfy the given guideline.

Terminating effect: $\{auth, auth\ access\}^{*} \cdot \{log\}$

Nonterminating effect: $\{auth, auth\ access\}^{\omega}$

Region Typing

- If a method was analyzed without considering object information, then its effect should include the traces of all objects.

  E.g., `y.last()` and `z.last()` would have the same effect.

- Then the terminating method `linear()` would have the same effect of the nonterminating method `cyclic()`.

- To improve the precision of effect typing, we use regions to narrow down referenced objects.

  Objects in different regions are analyzed separately.

```
class Node {
  Node next;
  Node last() {
    emit(a);
    if (next == null) {
      return this;
    } else {
      return next.last();
    }
  }
}

Class Test {
  Node linear() {
    Node x = new Node();
    Node y = new Node();
    y.next = x;
    return y.last();
  }
  Node cyclic() {
    Node z = new Node();
    z.next = z;
    return z.last();
  }
}
```

Region Type Systems for Featherweight Java

- **A pure region type system**

  [BGH13] L. Beringer, R. Grabowski, and M. Hofmann. **Verifying Pointer and String Analyses with Region Type Systems**. *Computer Languages, Systems & Structures* 39(2), 49–65, 2013.

- **A region-based effect type system (for analyzing terminating behaviors)**

  [EHZ17] S. Erbatur, M. Hofmann, and E. Zălinescu. **Enforcing Programming Guidelines with Region Types and Effects**. *APLAS 2017*.

- **Büchi effects (abstract interpretation based on Büchi automata)**

  [HC14] M. Hofmann and W. Chen. **Abstract Interpretation from Büchi Automata**. *CSL-LICS 2014*.

- **Another region-based effect type system (nonterminating and exceptional behaviors)**

  [ESX21] S. Erbatur, U. Schöpp, and C. Xu. **Type-based Enforcement of Infinitary Trace Properties for Java**. Preprint, 2021.

Goal: unify the above systems

- ❖ Relate and compare to other frameworks
- ❖ Extend to cover other language features
- ❖ Avoid redundant work on the meta theory

Featherweight Java (FJ)

- **Four kinds of names**

  variables: $x, y \in \mathrm{Var}$ classes: $C, D \in \mathrm{Cls}$ fields: $f \in \mathrm{Fld}$ methods: $m \in \mathrm{Mtd}$

- **Special formal elements**

  $\mathrm{this} \in \mathrm{Var}$ $\mathrm{Object}, \mathrm{NullType} \in \mathrm{Cls}$

- **FJ expressions**

$$\begin{aligned}\mathrm{Expr} \ni e \ ::= \ & x \mid \mathrm{let}\, x = e_1 \,\mathrm{in}\, e_2 \mid \mathrm{if}\, x = y \,\mathrm{then}\, e_1 \ \mathrm{else}\, e_2 \mid \mathrm{null} \mid \mathrm{new}^{\ell}\, C \mid (C)\, e \\ & \mid x^C.f \mid x^C.f \coloneqq y \mid x^C.m(\bar{y})\end{aligned}$$

- **An FJ program** $(\prec, \mathrm{fields}, \mathrm{methods}, \mathrm{mtable})$ consists of
  - a subtyping relation $\prec \in \mathcal{P}^{\mathrm{fin}}(\mathrm{Cls} \times \mathrm{Cls})$
  - a field list $\mathrm{fields} : \mathrm{Cls} \to \mathcal{P}^{\mathrm{fin}}(\mathrm{Fld})$
  - a method list $\mathrm{methods} : \mathrm{Cls} \to \mathcal{P}^{\mathrm{fin}}(\mathrm{Mtd})$
  - a method table $\mathrm{mtable} : \mathrm{Cls} \times \mathrm{Mtd} \rightharpoonup \mathrm{Var}^* \times \mathrm{Expr}$

Example of an FJ program

- **Java code**

```java
class Node {
  Node next;
  Node last() {
    emit(a);
    if (next == null) {
      return this;
    } else {
      return next.last();
    }
  }
}
```

- **FJ program**

$$\begin{aligned}
\mathrm{fields}(\mathrm{Node}) &= \{\mathrm{next}\} \\
\mathrm{methods}(\mathrm{Node}) &= \{\mathrm{last}\} \\
\mathrm{mtable}(\mathrm{Node}, \mathrm{last}) &= \big((\,)\,, e_{\mathrm{last}}\big)
\end{aligned}$$

$$\begin{aligned}
e_{\mathrm{last}} \coloneqq\; & \mathrm{let}\ _ = \mathrm{emit}(a)\ \mathrm{in} \\
& \mathrm{let}\ x = \mathrm{this.next}\ \mathrm{in} \\
& \mathrm{let}\ y = \mathrm{null}\ \mathrm{in} \\
& \mathrm{if}\ x = y\ \mathrm{then\ this} \\
& \mathrm{else\ let}\ z = \mathrm{this.next}\ \mathrm{in}\ z.\mathrm{last}()
\end{aligned}$$

A Parametric Operational Semantics

- **State model**

locations: $l \in \mathrm{Loc}$ stores: $s \in \mathrm{Var} \rightharpoonup \mathrm{Val}$

values: $v \in \mathrm{Val} = \mathrm{Loc} \uplus \{\mathrm{null}\}$ heaps: $h \in \mathrm{Loc} \rightharpoonup \mathrm{Obj}$

objects: $(C, G, \ell) \in \mathrm{Obj} = \mathrm{Cls} \times (\mathrm{Fld} \rightharpoonup \mathrm{Val}) \times \mathrm{Pos}$

Write $\mathcal{V}$ to denote the set of pairs (v, h) of values and heaps.

- **Parameter:** a set $\mathcal{M}$ together with functions

$$\mathrm{return}_{\mathcal{M}} : \mathcal{V} \to \mathcal{M} \qquad \mathrm{bind}_{\mathcal{M}} : \mathcal{M} \times \mathcal{M} \to \mathcal{M} \qquad |-|_{\mathcal{M}} : \mathcal{M} \to \mathcal{V}$$

such that

$$|\mathrm{return}_{\mathcal{M}}(v, h)|_{\mathcal{M}} = (v, h) \quad \text{and} \quad |\mathrm{bind}_{\mathcal{M}}(m_1, m_2)|_{\mathcal{M}} = |m_1|_{\mathcal{M}} \text{ or } |m_2|_{\mathcal{M}}$$

- **Big-step relation** $(s, h) \vdash e \Downarrow m$

Intuition: In state (s, h) the expression e evaluates to the value v with the heap updated to h', where $(v, h') = |m|_{\mathcal{M}}$.

Operational Semantics Rules

$$\frac{}{(s,h) \vdash x \Downarrow \mathsf{return}_{\mathcal{M}}(s(x),h)} \qquad \frac{}{(s,h) \vdash \mathtt{null} \Downarrow \mathsf{return}_{\mathcal{M}}(null,h)}$$

$$\frac{(s,h) \vdash e_1 \Downarrow m_1 \quad (v_1,h_1) = |m_1|_{\mathcal{M}} \quad (s[x \mapsto v_1],h_1) \vdash e_2 \Downarrow m_2}{(s,h) \vdash \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2 \Downarrow \mathsf{bind}_{\mathcal{M}}(m_1,m_2)}$$

$$\frac{s(x) = s(y) \quad (s,h) \vdash e_1 \Downarrow m}{(s,h) \vdash \mathtt{if}\ x = y\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2 \Downarrow m} \qquad \frac{s(x) \neq s(y) \quad (s,h) \vdash e_2 \Downarrow m}{(s,h) \vdash \mathtt{if}\ x = y\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2 \Downarrow m}$$

$$\frac{l \notin \mathrm{dom}(h) \quad G = [f \mapsto null]_{f \in fields(C)}}{(s,h) \vdash \mathsf{new}^{\ell}\ C \Downarrow \mathsf{return}_{\mathcal{M}}(l, h[l \mapsto (C,G,\ell)])}$$

$$\frac{(s,h) \vdash e \Downarrow m \quad (v,h') = |m|_{\mathcal{M}} \quad \mathrm{classOf}_{h'}(v) \leq C}{(s,h) \vdash (C)\, e \Downarrow m}$$

$$\frac{s(x) = l \qquad h(l) = (\_, G, \_)}{(s,h) \vdash x.f \Downarrow \mathsf{return}_{\mathcal{M}}(G(f),h)} \qquad \frac{s(x) = l \quad h(l) = (D,G,\ell) \quad h' = h[l \mapsto (D, G[f \mapsto s(y)], \ell)]}{(s,h) \vdash x.f := y \Downarrow \mathsf{return}_{\mathcal{M}}(s(y),h')}$$

$$\frac{s(x) = l \quad h(l) = (D,\_,\_) \quad mtable(D,m) = (\bar{z},e) \quad ([\mathtt{this} \mapsto l] \cup [z_i \mapsto s(y_i)]_{i \in \{1,\ldots,|\bar{z}|\}}, h) \vdash e \Downarrow m}{(s,h) \vdash x.m(\bar{y}) \Downarrow m}$$

Instances of the Operational Semantics

- **Standard FJ operational semantics**

  Simply take $\mathcal{M} = \mathcal{V}$, and $\mathrm{return}_{\mathcal{M}}$ and $|-|_{\mathcal{M}}$ the identity, and $\mathrm{bind}_{\mathcal{M}}$ the second projection.

  E.g.
$$\frac{(s,h) \vdash e_1 \Downarrow v_1, h_1 \qquad (s[x \mapsto v_1], h_1) \vdash e_2 \Downarrow v_2, h_2}{(s,h) \vdash \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2 \Downarrow v_2, h_2}$$

- **Operational semantics with trace effects**

  Apply the writer monad $X \mapsto X \times \Sigma^*$, i.e., take $\mathcal{M} = \mathcal{V} \times \Sigma^*$ and

$$\mathrm{return}_{\mathcal{M}}(v,h) = \big((v,h),\varepsilon\big)$$
$$\mathrm{bind}_{\mathcal{M}}\Big((\_,w_1),\big((v_2,h_2),w_2\big)\Big) = \big((v_2,h_2),w_1w_2\big)$$
$$\big|\big((v,h),\_\big)\big|_{\mathcal{M}} = (v,h)$$

  E.g.
$$\frac{(s,h) \vdash e_1 \Downarrow v_1, h_1\ \&\ w_1 \qquad (s[x \mapsto v_1], h_1) \vdash e_2 \Downarrow v_2, h_2\ \&\ w_2}{(s,h) \vdash \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2 \Downarrow v_2, h_2\ \&\ w_1w_2}$$

- **Operational semantics for FJ extended with e.g. exceptions and probabilistic branching**

Region Types

- A **region** represents a property of a value such as its provenance information.

$$\text{Reg} \ni r, s \ ::= \ \text{Null} \mid \text{CreatedAt}(\ell) \mid \top \mid \bot \mid r \vee s \mid r \wedge s$$

- A **formal interpretation** of regions as a relation $(v, h) \vdash r$

$$\frac{}{(null, h) \vdash \texttt{Null}} \qquad \frac{h(l) = (C, G, \ell)}{(l, h) \vdash \texttt{CreatedAt}\,(\ell)} \qquad \frac{}{(v, h) \vdash \top} \qquad \frac{(v, h) \vdash r}{(v, h) \vdash r \vee s} \qquad \frac{(v, h) \vdash s}{(v, h) \vdash r \vee s} \qquad \frac{(v, h) \vdash r \quad (v, h) \vdash s}{(v, h) \vdash r \wedge s}$$

- The interpretation gives a **partial order** $\leq$ on regions

$r \leq s$ iff $(v, h) \vdash r$ implies $(v, h) \vdash s$ for all $(v, h) \in \mathcal{V}$.

- Regions form a **lattice** $(\text{Reg}\,, \ \leq\,, \ \vee\,, \ \wedge)$

A Generic Region Type System

- **Parameter:** a join-semilattice $(\mathcal{L}, \emptyset, \sqsubseteq, \sqcup)$ together with function

  $$\mathrm{return}_{\mathcal{L}} : \mathrm{Reg} \to \mathcal{L} \qquad \mathrm{bind}_{\mathcal{L}} : \mathcal{L} \times (\mathrm{Reg} \to \mathcal{L}) \to \mathcal{L} \qquad |{-}|_{\mathcal{L}} : \mathcal{L} \to \mathcal{P}(\mathrm{Reg})$$

  Idea: $\mathcal{L}$ may carry information of e.g. regions, effects or probabilities with various representations.

  The essential structure of a region type system for FJ is given by a monad on the region lattice.

- **Typing judgments** have the form $x_1 : r_1, \ldots, x_n : r_n \vdash e : T$ where $r_1 \in \mathrm{Reg}$ and $T \in \mathcal{L}$.

- A **class table** (F, M) consists of
  - a field typing $F : \mathrm{Cls} \times \mathrm{Reg} \times \mathrm{Fld} \rightharpoonup \mathrm{Reg}$, and
  - a method typing $M : \mathrm{Cls} \times \mathrm{Reg} \times \mathrm{Mtd} \times \mathrm{Reg}^* \rightharpoonup \mathcal{L}$

  satisfying some well-formedness conditions that reflect the subtyping properties of FJ.

- An FJ program is **well-typed** w.r.t. (F, M) if each method body has the type as specified in M,

  i.e. $\mathrm{this} : r, \bar{x} : \bar{s} \vdash e : T$ holds for any $(C, r, m, \bar{s})$ with $M(C, r, m, \bar{s}) = T$ and $\mathrm{mtable}(C, m) = (\bar{x}, e)$.

Typing Rules

$$\textsc{Bot}\ \frac{(x{:}\,\bot) \in \Gamma}{\Gamma \vdash e : \emptyset} \qquad \textsc{Sub}\ \frac{\Gamma \vdash e : T \qquad T \sqsubseteq T'}{\Gamma \vdash e : T'}$$

$$\textsc{Var}\ \frac{}{\Gamma,\, x{:}\,r \vdash x : \mathsf{return}_{\mathcal{L}}(r)} \qquad \textsc{Null}\ \frac{}{\Gamma \vdash \mathtt{null} : \mathsf{return}_{\mathcal{L}}(\mathtt{Null})}$$

$$\textsc{Let}\ \frac{\Gamma \vdash e_1 : T_1 \qquad \Gamma,\, x{:}\,r \vdash e_2 : f(r)\ \text{ for all } r \in |T_1|_{\mathcal{L}}}{\Gamma \vdash \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2 : \mathsf{bind}_{\mathcal{L}}(T_1, f)}$$

$$\textsc{If}\ \frac{\Gamma,\, x{:}\,r{\wedge}s,\, y{:}\,r{\wedge}s \vdash e_1 : T_1 \qquad \Gamma,\, x{:}\,r,\, y{:}\,s \vdash e_2 : T_2}{\Gamma,\, x{:}\,r,\, y{:}\,s \vdash \mathtt{if}\ x = y\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2 : T_1 \sqcup T_2}$$

$$\textsc{New}\ \frac{}{\Gamma \vdash \mathsf{new}^{\ell}\ C : \mathsf{return}_{\mathcal{L}}(\mathsf{CreatedAt}\,(\ell))} \qquad \textsc{Cast}\ \frac{\Gamma \vdash e : T}{\Gamma \vdash (D)\, e : T}$$

$$\textsc{Get}\ \frac{s = F(C, r, f)}{\Gamma,\, x{:}\,r \vdash x^{C}.f : \mathsf{return}_{\mathcal{L}}(s)} \qquad \textsc{Set}\ \frac{s \leq F(C, r, f)}{\Gamma,\, x{:}\,r,\, y{:}\,s \vdash x^{C}.f := y : \mathsf{return}_{\mathcal{L}}(s)}$$

$$\textsc{Call}\ \frac{T = M(C, r, m, \bar{s})}{\Gamma,\, x{:}\,r,\, \bar{y}{:}\,\bar{s} \vdash x^{C}.m(\bar{y}) : T}$$

A Uniform Soundness Theorem

- Lift $(v, h) \vdash r$ to typing environments Γ and field typing F:
  - $(s, h) \vdash \Gamma$ iff $(s(x), h) \vdash r$ for all $(x{:}r) \in \Gamma$
  - $h \vdash F$ iff $(G(f), h) \vdash F(C, r, f)$ for all $l \in \mathrm{dom}(h)$ with $h(l) = (C, G, \_)$ and for all r, f with $(C, r, f) \in \mathrm{dom}(F)$

  We write $(s, h) \vdash \Gamma, F$ to denote the conjunction of $(s, h) \vdash \Gamma$ and $h \vdash F$.

  It says that the state (for evaluating the program) satisfies the properties specified by the typing.

- **Last parameter** $\lhd \subseteq \mathcal{M} \times \mathcal{L}$ to relate the parameters $\mathcal{M}$ and $\mathcal{L}$

Soundness Theorem. Suppose $\lhd \subseteq \mathcal{M} \times \mathcal{L}$ preserves the structures on $\mathcal{M}$ and $\mathcal{L}$ in the following sense:

($\lhd$1) $m \lhd T$ and $T \sqsubseteq T'$ implies $m \lhd T'$,

($\lhd$2) $(v, h) \vdash r$ implies $\mathrm{return}_{\mathcal{M}}(v, h) \lhd \mathrm{return}_{\mathcal{L}}(r)$, and

($\lhd$3) if $m \lhd T$ and if $m' \lhd f(r)$ for all $r \in |T|_{\mathcal{L}}$ with $|m|_{\mathcal{M}} \vdash r$, then $\mathrm{bind}_{\mathcal{M}}(m, m') \lhd \mathrm{bind}_{\mathcal{L}}(T, f)$.

Given an FJ program that is well-type w.r.t. (F, M), for any s, h, e, m, Γ and T such that

$$(s, h) \vdash e \Downarrow m \quad \text{and} \quad \Gamma \vdash e : T \quad \text{and} \quad (s, h) \vdash \Gamma, F$$

we have $m \lhd T$ and $(s, h') \vdash \Gamma, F$ where $(\_, h') = |m|_{\mathcal{M}}$.

Instantiating the Type System

- To build a concrete type system,
  provide a join-semilattice $(\mathcal{L}, \emptyset, \sqsubseteq, \sqcup)$ with maps $\mathrm{return}_{\mathcal{L}}$, $\mathrm{bind}_{\mathcal{L}}$ and $|-|_{\mathcal{L}}$.
- To establish its soundness result,
  - instantiate the operational semantics, i.e., choosing a set $\mathcal{M}$ with maps $\mathrm{return}_{\mathcal{M}}$, $\mathrm{bind}_{\mathcal{M}}$ and $|-|_{\mathcal{M}}$
  - specify the relation $\lhd \subseteq \mathcal{M} \times \mathcal{L}$ and verify the conditions ($\lhd$1), ($\lhd$2) and ($\lhd$3).

Instance: a pure region type system [BGH13]

- Take $(\mathcal{L}, \emptyset, \sqsubseteq, \sqcup) = (\mathrm{Reg}, \bot, \leq, \vee)$ with

$$\mathrm{return}_{\mathcal{L}}(r) = r \qquad \mathrm{bind}_{\mathcal{L}}(r, f) = f(r) \qquad |r|_{\mathcal{L}} = \{r\}$$

$$\frac{\Gamma \vdash e_1 : r_1 \qquad \Gamma, x{:}\, r_1 \vdash e_2 : r_2}{\Gamma \vdash \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2 : r_2}$$

E.g. $\mathrm{let}\ x = \mathrm{if}\ \mathit{cond}\ \mathrm{then}\ \left(\mathrm{new}^{\ell_1}\ C\right)\ \mathrm{else}\ \left(\mathrm{new}^{\ell_2}\ D\right)\ \mathrm{in}\ x \ : \ \mathrm{CreatedAt}(\ell_1) \vee \mathrm{CreatedAt}(\ell_2)$

- Work with the standard FJ operational semantics ($\mathcal{M} = \mathcal{V}$), and take $(v, h) \lhd r$ to be $(v, h) \vdash r$.

Instance: a Region-based Effect Type System [EHZ17]

- Take $\mathcal{L} = \mathrm{Reg} \times \mathcal{P}(\Sigma^*)$ with the lattice structure defined componentwise

  $e : (r, U)$ expresses that the result value of e is in region r and the generated event trace is in U.

- The monad functions are define by

$$\mathsf{return}_{\mathcal{L}}(r) = (r, \{\varepsilon\})$$

$$\mathsf{bind}_{\mathcal{L}}((r, U), f) = (s, UV) \quad \text{where } (s, V) = f(r)$$

$$|(r, u)|_{\mathcal{L}} = \{r\}$$

  The let-rule can be equivalently formulated as

$$\frac{\Gamma \vdash e_1 : (r_1, U_1) \qquad \Gamma,\ x{:}\,r_1 \vdash e_2 : (r_2, U_2)}{\Gamma \vdash \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2 : (r_2, U_1U_2)}$$

  E.g. $\mathrm{let}\ x = \mathrm{if}\ cond\ \mathrm{then}\ \big(\mathrm{emit}(a);\ \mathrm{new}^{\ell_1}\ C\big)\ \mathrm{else}\ \big(\mathrm{new}^{\ell_2}\ D\big)\ \mathrm{in}\ \mathrm{emit}(b);\ x$

  has type $(\mathrm{CreatedAt}(\ell_1) \vee \mathrm{CreatedAt}(\ell_2), \{ab, b\})$.

- Work with the operational semantics with traces ($\mathcal{M} = \mathcal{V} \times \Sigma^*$),

  and define $\big((v, h), w\big) \lhd (r, U) \iff \big((v, h) \vdash r\big) \wedge (w \in U)$.

Instance: another Region-based Effect Type System [ESX21]

- Take $\mathcal{L}$ to be the set of **finite partial functions** from Reg to $\mathcal{P}(\Sigma^*)$.

  $e : r_1 \,\&\, U_1 \mid \cdots \mid r_n \,\&\, U_n$ expresses that the result of e is in region r_i and the trace is in U_i for some i.

- We need to define the lattice structure and the monad functions (omitted).

  The let-rule can be equivalently formulated as

$$\frac{\Gamma \vdash e_1 : r_1 \,\&\, U_1 \mid \cdots \mid r_n \,\&\, U_n \qquad \Gamma,\, x{:}\, r_i \vdash e_2 : T_i \ \text{ for } 1 \le i \le n}{\Gamma \vdash \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2 : \bigsqcup_{i=1}^{n} U_i \cdot T_i}$$

  E.g. $\mathrm{let}\ x = \mathrm{if}\ cond\ \mathrm{then}\ \big(\mathrm{emit}(a);\ \mathrm{new}^{\ell_1}\ C\big)\ \mathrm{else}\ \big(\mathrm{new}^{\ell_2}\ D\big)\ \mathrm{in}\ \mathrm{emit}(b);\ x$

  has type $\mathrm{CreatedAt}(\ell_1) \,\&\, \{ab\} \mid \mathrm{CreatedAt}(\ell_2) \,\&\, \{b\}$.

- Still work with the operational semantics with traces ($\mathcal{M} = \mathcal{V} \times \Sigma^*$),

  but define $\big((v,h),w\big) \lhd (r_1 \,\&\, U_1 \mid \cdots \mid r_n \,\&\, U_n) \iff \exists\, i. \big((v,h) \vdash r_i\big) \wedge (w \in U_i)$.

Comparing the Instances [EHZ17] and [ESX21]

Example: Suppose there are classes $D \prec C$ with two methods f and g.

Consider the class table:

$$\begin{array}{ll} \texttt{class}\ C\texttt{@CreatedAt}(\ell_1) & \texttt{class}\ D\texttt{@CreatedAt}(\ell_2) \\ \quad f() : \texttt{Null}\ \&\ \{a\} & \quad f() : \texttt{Null}\ \&\ \{b\} \\ \quad g() : \texttt{Null}\ \&\ \{aa\} & \quad g() : \texttt{Null}\ \&\ \{bb\} \end{array}$$

- Let e be the FJ expression $\quad \text{if}\ cond\ \text{then}\ \left(\text{new}^{\ell_1}\ C\right)\ \text{else}\ \left(\text{new}^{\ell_2}\ D\right)$
  - In [EHZ17], e has type $\text{CreatedAt}(\ell_1) \lor \text{CreatedAt}(\ell_2)\ \ \&\ \ \{\varepsilon\}$
  - In [ESX21], e has type $\text{CreatedAt}(\ell_1)\ \&\ \{\varepsilon\}\ \mid\ \text{CreatedAt}(\ell_2)\ \&\ \{\varepsilon\}$
- Consider expressions $\quad \text{let}\ x = e\ \text{in}\ x.f(); x.f() \quad$ and $\quad \text{let}\ x = e\ \text{in}\ x.g()$
  - In [EHZ17], the former has type $\text{null}\ \&\ \{aa, ab, ba, bb\}$ and the latter has $\text{null}\ \&\ \{aa, bb\}$
  - In [ESX21], both have type $\text{null}\ \&\ \{aa, bb\}$.
- The method g may have body $\text{this}.f(); \text{this}.f()$. Inlining loses precision in [EHZ17].
- [ESX21] is more precise, but the cost is a less efficient type inference algorithm.

Extension: Exception Handling

- Extend the syntax of FJ with expressions $\text{throw}\ e$ and $\text{try}\ e_1\ \text{catch}(C\ x)\ e_2$
- For the operational semantics, work with e.g. $\mathcal{M} = \{\mathrm{N}, \mathrm{E}\} \times \mathcal{V}$
  - $(s, h) \vdash e \Downarrow \mathrm{N}, v, h'$ means that e normalizes to v with the heap updated to h'.
  - $(s, h) \vdash e \Downarrow \mathrm{E}, v, h'$ means that e throws an exception whose value is v and the heap is updated to h'.
- The monad functions are given by

$$\begin{aligned}
\mathsf{return}_{\mathcal{M}}(v, h) &= (\mathsf{N}, (v, h)) \\
\mathsf{bind}_{\mathcal{M}}((\mathsf{N}, _), (x, (v, h))) &= (x, (v, h)) \\
\mathsf{bind}_{\mathcal{M}}((\mathsf{E}, (v, h)), _) &= (\mathsf{E}, (v, h)) \\
|(_, (v, h))|_{\mathcal{M}} &= (v, h).
\end{aligned}$$

- Think about all the possible cases of

$$\frac{(s, h) \vdash e_1 \Downarrow m_1 \qquad (v_1, h_1) = |m_1|_{\mathcal{M}} \qquad (s[x \mapsto v_1], h_1) \vdash e_2 \Downarrow m_2}{(s, h) \vdash \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2 \Downarrow \mathsf{bind}_{\mathcal{M}}(m_1, m_2)}$$

- Additional operational semantics rules for the new expressions such as

$$\frac{(s, h) \vdash e \Downarrow \_, v, h'}{(s, h) \vdash \mathtt{throw}\ e \Downarrow \mathsf{E}, v, h'}$$

Extension: Exception Handling (cont.)

- Extend the pure region type system [BGH13] by taking $\mathcal{L} = \mathrm{Reg} \times \mathrm{Reg}$

  $e : (r, s)$ says that e evaluates to a value in region r, or throws an exception whose value is in region s.

- The monad functions are define by

$$\mathrm{return}_{\mathcal{L}}(r) = (r, \bot)$$

$$\mathrm{bind}_{\mathcal{L}}((r,s), f) = (t, s \vee u) \quad \text{where } (t,u) = f(r)$$

$$|(r,s)|_{\mathcal{L}} = \{r\}$$

  The let-rule can be equivalently formulated as

$$\textsc{let}\ \frac{\Gamma \vdash e_1 : (r_1, s_1) \qquad \Gamma,\ x{:}r_1 \vdash e_2 : (r_2, s_2)}{\Gamma \vdash \texttt{let}\ x = e_1\ \texttt{in}\ e_2 : (r_2, s_1 \vee s_2)}$$

- Additional typing rules for the new expressions such as

$$\textsc{throw}\ \frac{\Gamma \vdash e : (r, s)}{\Gamma \vdash \texttt{throw}\ e : (\bot, r \vee s)}$$

- Lastly, define $\lhd$ by $\big(\mathrm{N}, (v,h)\big) \lhd (r,s) \iff (v,h) \vdash r$ and $\big(\mathrm{E}, (v,h)\big) \lhd (r,s) \iff (v,h) \vdash s$

- Once ($\lhd$1)—($\lhd$3) are verified, the soundness theorem is valid for the core FJ calculus, we only need to prove the cases for the additional rules.

Extension: Probabilistic Branching (w.i.p.)

- Extend the syntax of FJ with $e_1 \; ?^p \; e_2$ where $p \in [0,1]$ is the probability of evaluating to the left.

  Goal: use the type system to compute the probability of a program generating given traces.

- For the operational semantics, work with $\mathcal{M} = [0,1] \times \mathcal{V} \times \Sigma^*$

  $(s, h) \vdash e \Downarrow_p v, h' \;\&\; w$ means that e has a probability p of evaluating to v with the heap updated to h' and generating the event trace w.

- Additional operational semantics rules for $?^p$

$$\frac{(s,h) \vdash e_1 \Downarrow_p v, h' \;\&\; w}{(s,h) \vdash e_1 ?^q e_2 \Downarrow_{p \times q} v, h' \;\&\; w} \qquad \frac{(s,h) \vdash e_2 \Downarrow_p v, h' \;\&\; w}{(s,h) \vdash e_1 ?^q e_2 \Downarrow_{p \times (1-q)} v, h' \;\&\; w}$$

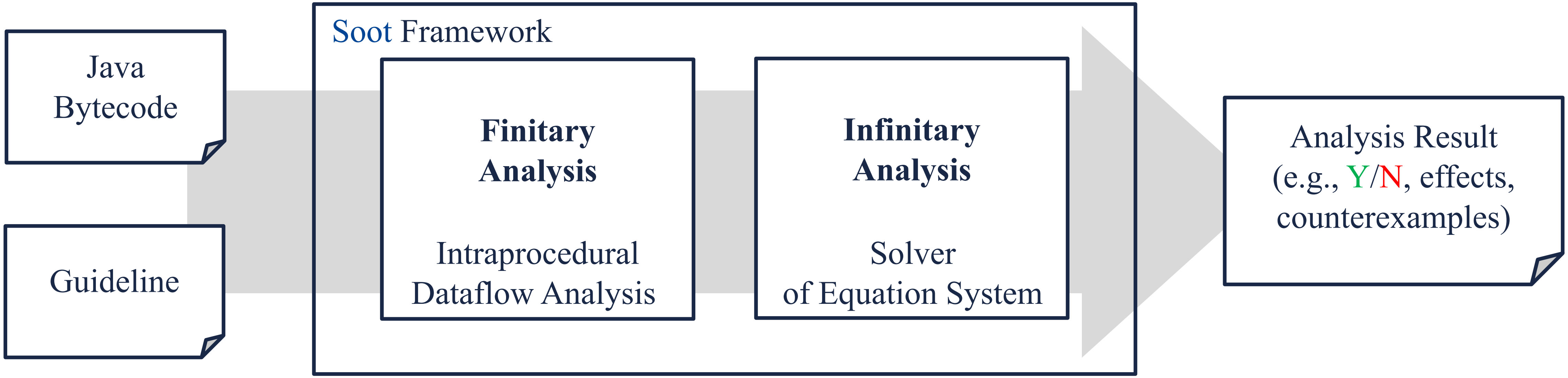
- For the type system, take $\mathcal{L} = \mathrm{Reg} \times (\mathcal{P}(\Sigma^*) \to [0,1])$

  $e : (r, \theta)$ says that the result of e is in region r, and it has a probability function θ such that $\theta(U)$ is the probability of e generating traces of U.

- Additional typing rules for $?^p$

$$\frac{\Gamma \vdash e_1 : (r_1, \theta_1) \qquad \Gamma \vdash e_2 : (r_2, \theta_2)}{\Gamma \vdash e_1 ?^p e_2 : (r_1 \vee r_2,\; p \cdot \theta_1 + (1-p) \cdot \theta_2)}$$

Prototype Implementation



A prototype implementation of type inference based on the Soot framework:

- Effects are represented by the finitary abstraction based on the guideline automaton.
- The guideline also specifies the default effects of intrinsic functions.
- For libraries, we assume default effects or provide mockup code.

Summary

- We introduce a generic region type system for FJ and prove a uniform soundness theorem.
- It unifies the systems investigated in the GuideForce project.
- The uniform framework is helpful when extending FJ to cover other language features.
- This talk is based on the following paper

U. Schöpp and C. Xu. **A Generic Region Type System for Featherweight Java**. To appear at *FTfJP 2021*.

Thank you!