A relational proof system for information flow security of unstructured bytecode

Lennart Beringer

Lehrstuhl für Theoretische Informatik
Ludwig-Maximilians-Universität München

IoC Tallinn, December 11th, 2007

What is information flow security?

Design choice: object identity is oberservable (I do that) vs extensional object identity (equality of dynamic class and fields)

Informal definition

A system is called (information flow) secure if an outside attacker cannot obtain knowledge about internal secret information by interacting with the system (i.e. by repeatedly applying some input and observing the output).

Precise notions of terms attacker, public/private, information, system vary.

Concrete notions exist for various modeling frameworks:

- Automata/ state machines (Goguen/Meseguer, Rushby)
- Process calculi
- Programming languages

Information flow analysis in programming languages

- Classify data (or variables, objects, locations,...) according to security levels L/H (or lattice)
- Non-interference: initial differences in high data are not semantically observable at the low domain
- Example (imperative language):
  $C \ secure \iff \forall \ s \ s'. \ s =_L s' \Rightarrow [\![C]\!]s \approx_L [\![C]\!]s'$
- Various possibilities for semantics $[\![C]\!]$: terminal states, termination sensitive...
- Examples for $s =_L s' \iff s(l) = s'(l)$ and
  $[\![C]\!]s \approx_L [\![C]\!]s' \iff \forall \ t \ t'. \ s \xrightarrow{C} t \Rightarrow s' \xrightarrow{C} t' \Rightarrow t =_L t'$:
  - $l{:=}h$ (insecure; direct flow) vs. $h := l$ (secure)
  - **if** $h = 0$ **then** $l{:=}1$ **else** $l{:=}2$ (insecure; indirect flow)
  - **if** $h = 0$ **then** $l{:=}1$ **else** $l{:=}1$ and $l{:=}h; l{:=}1$ (secure, but rejected by many (static) verification techniques)

Static analysis: type system by Volpano/Smith/Irvine

- Types τ = security levels (*high*/*low*), with $low \sqsubset high$
- Typing rules prevent assignments $l := e$ if e depends on a high variable, or if a surrounding conditional (or loop guard) depends on a high boolean expression
- Γ associates (fixed) types to variables
- Typing judgements:
  - $\vdash e : \tau$: τ is a upper bound on the variables occurring in e, i.e. e does not depend on any variable x with $\Gamma(x) \sqsupset \tau$
  - $\tau \vdash C$: τ is an lower bound on the variables assigned to in C, i.e. variables x with $\Gamma(x) \sqsubset \tau$ remain unchanged

$$\frac{\vdash e : \tau \quad \Gamma(x) = \tau}{\tau \vdash x{:=}e} \qquad \frac{\tau \vdash C \quad \tau \vdash D}{\tau \vdash C; D} \qquad \frac{\vdash b : \tau \quad \tau \vdash C_i}{\tau \vdash \textbf{if } b \textbf{ then } C_1 \textbf{ else } C_2}$$

. . . plus subtyping rules.

Theorem

If $low \vdash C$ then C is secure.

Evaluation of Volpano/Smith/Irvine

- Soundness result ensures that programs with direct (example $l{:=}h$) or indirect flows (example **if** $h = 0$ **then** $l{:=}1$ **else** $l{:=}2$) are eliminated, while e.g. $h := l$ is rightly admitted
- Shortcoming I: subprograms of well-typed programs are required to be secure, e.g. $l{:=}h; l{:=}1$ is rejected
- Shortcoming II: no direct compatibility with program transformations. E.g. **if** $h = 0$ **then** $l{:=}1$ **else** $l{:=}1$ and **if** $h = 0$ **then** $l{:=}1; x{:=}2$ **else** $x{:=}2; l{:=}1$ where $x : low$ are rejected
- Generalisation to flow-sensitivity (Hunt & Sands) removes some of the shortcomings

Our aim

Present better type system, for low-level language (PCC motivation)

Non-interference for bytecode

Bytecode: low-level (virtual machine level) programming language

- Components of states: operand stack, store, heap
- Code: load/store between store and operand stack, arithmetic operations, object creation and manipulation, method calls, unstructured control flow (concurrency, exceptions,...)

Current type systems are essentially bytecode-level versions of VSI:

- require essentially structured control flow: employ a *pc*-type in order to ensure no low assignments (or method returns) under a high branch discipline (or CDR,...)
- cannote directly exploit if a program transformations preserve non-interference

Goal: present a proof system for unstructured bytecode that does not flatly reject all low assignments (or returns) in high branches, and is compatible with program transformations

Stack-based language

- Simple programs: no heap, no methods.
- Big-step operational semantics: $\mathcal{P} \vdash ([\,], \sigma), \ell \Downarrow v$

(Termination-insensitive) Non-interference for bytecode

Program label ℓ is non-interferent for $\mathcal{S} : \mathcal{X} \to \mathcal{L}$, if $\mathcal{P} \vdash ([\,], \sigma), \ell \Downarrow v$ and $\mathcal{P} \vdash ([\,], \tau), \ell \Downarrow w$ implies $v \sim_{low} w$ whenever $\sigma \sim_{\mathcal{S}} \tau$.

Idea

1. Non-interference is a special case of similarity:

Similarity

Labels ℓ and ℓ' are (low) similar for $\alpha \in \mathcal{L}^*$ and $\mathcal{S}$, notation $\ell \sim_{(\alpha,\mathcal{S})} \ell'$, if $\mathcal{P} \vdash (\mathcal{O}, \sigma), \ell \Downarrow v$ and $\mathcal{P} \vdash (\mathcal{O}', \tau), \ell' \Downarrow w$ implies $v \sim_{low} w$ whenever $\sigma \sim_{\mathcal{S}} \tau$ and $\mathcal{O} \sim_\alpha \mathcal{O}'$.

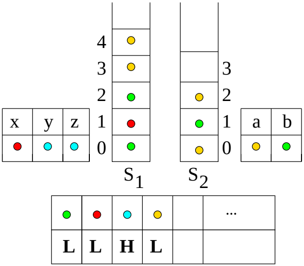
2. Generalise pairs $(\alpha, \mathcal{S})$ to *relational shape descriptions* β, such that identity of values may be traced (not: formal variable dependencies) through both computations
3. Derive *relational proof system* where abstractions β play the role of types (applying to the initial states)

Relational shape descriptions (RSD's)

$\mathcal{C}$: infinite set of colours, ranged over by γ.

Definition

A RSD is a structure $\beta = ((S, \Gamma), N, (T, \Delta))$ where $S, T \in \mathcal{C}^*$, $\Gamma, \Delta \in \mathcal{X} \rightharpoonup_{fin} \mathcal{C}$ and $N \in \mathcal{C} \rightharpoonup_{fin} \mathcal{L}$, such that $cod\ S \cup cod\ T \cup cod\ \Gamma \cup cod\ \Delta \subseteq dom\ N$.



Interpretation of RSD's (informal)

Definition

The interpretation of γ in a state $\mathcal{O}, \sigma$ with respect to (S, Γ) (where $|\mathcal{O}| = |S|$) is the set of values at the positions containing γ:

$$[\![\gamma]\!]_{(S,\Gamma)}((\mathcal{O}, \sigma)) = \{\mathcal{O}(n) \mid S(n) = \gamma\} \cup \{\sigma(x) \mid \Gamma(x) = \gamma\}$$

Definition

States $(\mathcal{O}_1, \sigma)$ and $(\mathcal{O}_2, \tau)$ are indistinguishable with respect to β, notation $(\mathcal{O}_1, \sigma) \sim_\beta (\mathcal{O}_2, \tau)$, if for all $\gamma \in \mathit{dom}\ N$,

- $[\![\gamma]\!]_{(S,\Gamma)}((\mathcal{O}, \sigma))$ and $[\![\gamma]\!]_{(T,\Delta)}((\mathcal{O}', \sigma'))$ are either empty or singleton sets
- If $[\![\gamma]\!]_{(S,\Gamma)}((\mathcal{O}, \sigma)) = \{v\}$ and $[\![\gamma]\!]_{(T,\Delta)}((\mathcal{O}', \sigma')) = \{w\}$ and $N(\gamma) = \mathit{low}$ then $v = w$

Similarity

Definition

Code points ℓ and ℓ' are (β, p)-similar if $v \sim_p w$ holds whenever $\mathcal{P} \vdash s, \ell \Downarrow v$ and $\mathcal{P} \vdash t, \ell' \Downarrow w$ and $s \sim_\beta t$.

Now derive proof system for (β, p)-similarity.

- one-side rules (high rules, or propagation of identifiers), incl. rule of symmetry
- two-sided rules (low rules)
- structural rules (axioms, inject, subtyping)

One-sided rules

ι	$\beta(\iota)$	$\beta'(\iota)$	$\Phi(\iota)$
load x	$((S,\Gamma),N,(T,\Delta))$	$((\gamma :: S,\Gamma),N,(T,\Delta))$	$\Gamma{\downarrow}x = \gamma$
store x	$((\gamma :: S,\Gamma),N,(T,\Delta))$	$((S,\Gamma[x \mapsto \gamma]),N,(T,\Delta))$	
pop	$((\gamma :: S,\Gamma),N,(T,\Delta))$	$((S,\Gamma),N,(T,\Delta))$	
dup	$((\gamma :: S,\Gamma),N,(T,\Delta))$	$((\gamma :: \gamma :: S,\Gamma),N,(T,\Delta))$	
swap	$((\gamma_1 :: \gamma_2 :: S,\Gamma),N,(T,\Delta))$	$((\gamma_2 :: \gamma_1 :: S,\Gamma),N,(T,\Delta))$	
iconst v	$((S,\Gamma),N,(T,\Delta))$	$((\gamma :: S,\Gamma),N[\gamma \mapsto q],(T,\Delta))$	$\gamma \notin dom\ N$
binop $\oplus$	$((\gamma_1 :: \gamma_2 :: S,\Gamma),N,(T,\Delta))$	$((\gamma :: S,\Gamma),N[\gamma \mapsto q],(T,\Delta))$	$\begin{cases} N{\downarrow}\gamma_2 = q_2 \\ N{\downarrow}\gamma_1 = q_1 \\ \gamma \notin dom\ N \\ q = q_1 \sqcup q_2 \end{cases}$

$$\textsc{RInstr}\ \frac{M(\ell) = \iota \quad \Phi(\iota) \quad G \vdash_t \ell+1 * \ell' : \beta'(\iota) \longrightarrow p}{G \vdash_f \ell * \ell' : \beta(\iota) \longrightarrow p}$$

$$\textsc{RGoto}\ \frac{M(\ell) = \text{goto}\ \bar{\ell} \quad G \vdash_t \bar{\ell} * \ell' : \beta \longrightarrow p}{G \vdash_f \ell * \ell' : \beta \longrightarrow p} \qquad \textsc{RSymm}\ \frac{G^{-1} \vdash_b \ell' * \ell : \beta^{-1} \longrightarrow p}{G \vdash_b \ell * \ell' : \beta \longrightarrow p}$$

$$\textsc{RIf}\ \frac{M(\ell) = \text{iftrue}\ \bar{\ell} \qquad \begin{array}{c} G \vdash_t \ell+1 * \ell' : ((S,\Gamma),N,(T,\Delta)) \longrightarrow p \\ G \vdash_t \bar{\ell} * \ell' : ((S,\Gamma),N,(T,\Delta)) \longrightarrow p \end{array}}{G \vdash_f \ell * \ell' : ((\gamma :: S,\Gamma),N,(T,\Delta)) \longrightarrow p}$$

One-sided rules

ι	$\beta(\iota)$	$\beta'(\iota)$	$\Phi(\iota)$
load x	$((S,\Gamma),N,(T,\Delta))$	$((\gamma :: S,\Gamma),N,(T,\Delta))$	$\Gamma{\downarrow}x = \gamma$
store x	$((\gamma :: S,\Gamma),N,(T,\Delta))$	$((S,\Gamma[x \mapsto \gamma]),N,(T,\Delta))$	
pop	$((\gamma :: S,\Gamma),N,(T,\Delta))$	$((S,\Gamma),N,(T,\Delta))$	
dup	$((\gamma :: S,\Gamma),N,(T,\Delta))$	$((\gamma :: \gamma :: S,\Gamma),N,(T,\Delta))$	
swap	$((\gamma_1 :: \gamma_2 :: S,\Gamma),N,(T,\Delta))$	$((\gamma_2 :: \gamma_1 :: S,\Gamma),N,(T,\Delta))$	
iconst v	$((S,\Gamma),N,(T,\Delta))$	$((\gamma :: S,\Gamma),N[\gamma \mapsto q],(T,\Delta))$	$\gamma \notin dom\ N$
binop $\oplus$	$((\gamma_1 :: \gamma_2 :: S,\Gamma),N,(T,\Delta))$	$((\gamma :: S,\Gamma),N[\gamma \mapsto q],(T,\Delta))$	$\begin{cases} N{\downarrow}\gamma_2 = q_2 \\ N{\downarrow}\gamma_1 = q_1 \\ \gamma \notin dom\ N \\ q = q_1 \sqcup q_2 \end{cases}$

$$\text{RInstr}\ \frac{M(\ell) = \iota \quad \Phi(\iota) \quad G \vdash_t \ell+1 * \ell' : \beta'(\iota) \to p}{G \vdash_f \ell * \ell' : \beta(\iota) \to p}$$

$$\text{RGoto}\ \frac{M(\ell) = \text{goto}\ \bar{\ell} \quad G \vdash_t \bar{\ell} * \ell' : \beta \to p}{G \vdash_f \ell * \ell' : \beta \to p} \qquad \text{RSymm}\ \frac{G^{-1} \vdash_b \ell' * \ell : \beta^{-1} \to p}{G \vdash_b \ell * \ell' : \beta \to p}$$

$$\text{RIf}\ \frac{M(\ell) = \text{iftrue}\ \bar{\ell} \quad \begin{array}{r} G \vdash_t \ell+1 * \ell' : ((S,\Gamma),N,(T,\Delta)) \to p \\ G \vdash_t \bar{\ell} * \ell' : ((S,\Gamma),N,(T,\Delta)) \to p \end{array}}{G \vdash_f \ell * \ell' : ((\gamma :: S,\Gamma),N,(T,\Delta)) \to p}$$

Two-sided and structural rules

$$\textsc{RConstConst}\ \frac{M(\ell) = \text{iconst } v \qquad M'(\ell') = \text{iconst } w \qquad v \sim_q w \qquad \gamma \notin dom\ N \qquad G \vdash_t \ell + 1 * \ell' + 1 : ((\gamma : S, \Gamma), N[\gamma \mapsto q], (\gamma :: T, \Delta)) \to p}{G \vdash_f \ell * \ell' : ((S, \Gamma), N, (T, \Delta)) \to p}$$

$$\textsc{RRetRetH}\ \frac{M(\ell) = \text{vreturn} \qquad M'(\ell') = \text{vreturn}}{G \vdash_f \ell * \ell' : ((\gamma_1 :: S, \Gamma), N, (\gamma_2 :: T, \Delta)) \to high}$$

$$\textsc{RRetRet}\ \frac{M(\ell) = \text{vreturn} \qquad M'(\ell') = \text{vreturn} \qquad N{\downarrow}\gamma = p}{G \vdash_f \ell * \ell' : ((\gamma :: S, \Gamma), N, (\gamma :: T, \Delta)) \to p}$$

$$\textsc{RIfIf}\ \frac{M(\ell) = \text{iftrue } \bar{\ell} \qquad M'(\ell') = \text{iftrue } \bar{\ell}_1 \qquad N{\downarrow}\gamma = low \qquad G \vdash_t \ell + 1 * \ell' + 1 : ((S, \Gamma), N, (T, \Delta)) \to p \qquad G \vdash_t \bar{\ell} * \bar{\ell}_1 : ((S, \Gamma), N, (T, \Delta)) \to p}{G \vdash_f \ell * \ell' : ((\gamma :: S, \Gamma), N, (\gamma :: T, \Delta)) \to p}$$

$$\textsc{Ax}\ \frac{G{\downarrow}(\ell, \ell') = (\beta, p)}{G \vdash_t \ell * \ell' : \beta \to p} \qquad \textsc{Sub}\ \frac{G \vdash_b \ell * \ell' : \beta \to q \qquad \beta <: \beta' \qquad q \sqsubseteq p}{G \vdash_b \ell * \ell' : \beta' \to p}$$

Two-sided and structural rules

$$\textsc{RConstConst}\ \frac{\begin{array}{c}M(\ell) = \mathrm{iconst}\ v \qquad M'(\ell') = \mathrm{iconst}\ w \\ v \sim_q w \qquad \gamma \notin dom\ N \\ G \vdash_{\mathrm{t}} \ell+1 * \ell'+1 : ((\gamma : S, \Gamma), N[\gamma \mapsto q], (\gamma :: T, \Delta)) \rightarrow p\end{array}}{G \vdash_{\mathrm{f}} \ell * \ell' : ((S, \Gamma), N, (T, \Delta)) \rightarrow p}$$

$$\textsc{RRetRetH}\ \frac{M(\ell) = \mathrm{vreturn} \qquad M'(\ell') = \mathrm{vreturn}}{G \vdash_{\mathrm{f}} \ell * \ell' : ((\gamma_1 :: S, \Gamma), N, (\gamma_2 :: T, \Delta)) \rightarrow high}$$

$$\textsc{RRetRet}\ \frac{M(\ell) = \mathrm{vreturn} \quad M'(\ell') = \mathrm{vreturn} \quad N{\downarrow}\gamma = p}{G \vdash_{\mathrm{f}} \ell * \ell' : ((\gamma :: S, \Gamma), N, (\gamma :: T, \Delta)) \rightarrow p}$$

$$\textsc{RIfIf}\ \frac{\begin{array}{c}M(\ell) = \mathrm{iftrue}\ \bar{\ell} \quad M'(\ell') = \mathrm{iftrue}\ \bar{\ell}_1 \quad N{\downarrow}\gamma = low \\ G \vdash_{\mathrm{t}} \ell+1 * \ell'+1 : ((S, \Gamma), N, (T, \Delta)) \rightarrow p \\ G \vdash_{\mathrm{t}} \bar{\ell} * \bar{\ell}_1 : ((S, \Gamma), N, (T, \Delta)) \rightarrow p\end{array}}{G \vdash_{\mathrm{f}} \ell * \ell' : ((\gamma :: S, \Gamma), N, (\gamma :: T, \Delta)) \rightarrow p}$$

$$\textsc{Ax}\ \frac{G{\downarrow}(\ell, \ell') = (\beta, p)}{G \vdash_{\mathrm{t}} \ell * \ell' : \beta \rightarrow p} \qquad \textsc{Sub}\ \frac{G \vdash_b \ell * \ell' : \beta \rightarrow q \quad \beta <: \beta' \quad q \sqsubseteq p}{G \vdash_b \ell * \ell' : \beta' \rightarrow p}$$

Transformation rules

Derivable rules, e.g.

$$\frac{M(\ell) = \text{store } x \quad M(\ell+1) = \text{store } y \quad M'(\ell') = \text{store } y \quad M'(\ell'+1) = \text{store } x \\ G \vdash_{\mathrm{t}} \ell + 2 * \ell' + 2 : ((S, \Gamma[x \mapsto \gamma_1][y \mapsto \gamma_2]), N, (T, \Delta[y \mapsto \gamma_3][x \mapsto \gamma_4])) \to p}{G \vdash_{\mathrm{f}} \ell * \ell' : ((\gamma_1 :: \gamma_2 :: S, \Gamma), N, (\gamma_3 :: \gamma_4 :: T, \Delta)) \to p}$$

Admissible rules, e.g.

$$\frac{\begin{array}{c} M(\ell) = \text{iconst } v \quad M(\ell+1) = \text{store } x \quad M(\ell+2) = \text{iconst } w \quad M(\ell+3) = \text{store } y \\ M'(\ell') = \text{iconst } w \quad M'(\ell'+1) = \text{store } y \\ M'(\ell'+2) = \text{iconst } v \quad M'(\ell'+3) = \text{store } x \\ \gamma_1 \neq \gamma_2 \quad \{\gamma_1, \gamma_2\} \cap dom\ N = \emptyset \quad G \vdash_{\mathrm{t}} \ell + 4 * \ell' + 4 : \beta \to p \\ \beta = ((S, \Gamma[x \mapsto \gamma_1][y \mapsto \gamma_2]), N[\gamma_1 \mapsto q_1][\gamma_2 \mapsto q_2], (T, \Delta[y \mapsto \gamma_2][x \mapsto \gamma_1])) \end{array}}{G \vdash_{\mathrm{f}} \ell * \ell' : ((S, \Gamma), N, (T, \Delta)) \to p}$$

Rule of transitivity:

$$\frac{\begin{array}{cc} G \vdash_{(\mathrm{f},\mathrm{f})} \ell * \ell' : ((S, \Gamma), N, (T, \Delta)) \to p & \rhd G \\ G \vdash_{(\mathrm{f},\mathrm{f})} \ell' * \ell'' : ((T, \Delta), N, (R, \Sigma)) \to p & dom\ T \cup cod\ \Delta \subseteq dom\ N \end{array}}{G \vdash_{(\mathrm{f},\mathrm{f})} \ell * \ell'' : ((S, \Gamma), N, (R, \Sigma)) \to p}$$

Transformation rules

Derivable rules, e.g.

$$\frac{M(\ell) = \text{store } x \quad M(\ell+1) = \text{store } y \quad M'(\ell') = \text{store } y \quad M'(\ell'+1) = \text{store } x \qquad G \vdash_{\mathrm{t}} \ell+2 * \ell'+2 : ((S, \Gamma[x \mapsto \gamma_1][y \mapsto \gamma_2]), N, (T, \Delta[y \mapsto \gamma_3][x \mapsto \gamma_4])) \to p}{G \vdash_{\mathrm{f}} \ell * \ell' : ((\gamma_1 :: \gamma_2 :: S, \Gamma), N, (\gamma_3 :: \gamma_4 :: T, \Delta)) \to p}$$

Admissible rules, e.g.

$$\frac{\begin{array}{c} M(\ell) = \text{iconst } v \quad M(\ell+1) = \text{store } x \quad M(\ell+2) = \text{iconst } w \quad M(\ell+3) = \text{store } y \\ M'(\ell') = \text{iconst } w \quad M'(\ell'+1) = \text{store } y \\ M'(\ell'+2) = \text{iconst } v \quad M'(\ell'+3) = \text{store } x \\ \gamma_1 \neq \gamma_2 \quad \{\gamma_1, \gamma_2\} \cap dom\ N = \emptyset \quad G \vdash_{\mathrm{t}} \ell+4 * \ell'+4 : \beta \to p \\ \beta = ((S, \Gamma[x \mapsto \gamma_1][y \mapsto \gamma_2]), N[\gamma_1 \mapsto q_1][\gamma_2 \mapsto q_2], (T, \Delta[y \mapsto \gamma_2][x \mapsto \gamma_1])) \end{array}}{G \vdash_{\mathrm{f}} \ell * \ell' : ((S, \Gamma), N, (T, \Delta)) \to p}$$

Rule of transitivity:

$$\frac{G \vdash_{(\mathrm{f},\mathrm{f})} \ell * \ell' : ((S, \Gamma), N, (T, \Delta)) \to p \qquad \triangleright G \\ G \vdash_{(\mathrm{f},\mathrm{f})} \ell' * \ell'' : ((T, \Delta), N, (R, \Sigma)) \to p \qquad dom\ T \cup cod\ \Delta \subseteq dom\ N}{G \vdash_{(\mathrm{f},\mathrm{f})} \ell * \ell'' : ((S, \Gamma), N, (R, \Sigma)) \to p}$$

Transformation rules

Derivable rules, e.g.

$$\frac{M(\ell) = \text{store } x \quad M(\ell+1) = \text{store } y \quad M'(\ell') = \text{store } y \quad M'(\ell'+1) = \text{store } x \\ G \vdash_{\mathrm{t}} \ell + 2 * \ell' + 2 : ((S, \Gamma[x \mapsto \gamma_1][y \mapsto \gamma_2]), N, (T, \Delta[y \mapsto \gamma_3][x \mapsto \gamma_4])) \rightarrow p}{G \vdash_{\mathrm{f}} \ell * \ell' : ((\gamma_1 :: \gamma_2 :: S, \Gamma), N, (\gamma_3 :: \gamma_4 :: T, \Delta)) \rightarrow p}$$

Admissible rules, e.g.

$$\frac{\begin{array}{c} M(\ell) = \text{iconst } v \quad M(\ell+1) = \text{store } x \quad M(\ell+2) = \text{iconst } w \quad M(\ell+3) = \text{store } y \\ M'(\ell') = \text{iconst } w \quad M'(\ell'+1) = \text{store } y \\ M'(\ell'+2) = \text{iconst } v \quad M'(\ell'+3) = \text{store } x \\ \gamma_1 \neq \gamma_2 \quad \{\gamma_1, \gamma_2\} \cap dom\ N = \emptyset \quad G \vdash_{\mathrm{t}} \ell + 4 * \ell' + 4 : \beta \rightarrow p \\ \beta = ((S, \Gamma[x \mapsto \gamma_1][y \mapsto \gamma_2]), N[\gamma_1 \mapsto q_1][\gamma_2 \mapsto q_2], (T, \Delta[y \mapsto \gamma_2][x \mapsto \gamma_1])) \end{array}}{G \vdash_{\mathrm{f}} \ell * \ell' : ((S, \Gamma), N, (T, \Delta)) \rightarrow p}$$

Rule of transitivity:

$$\frac{\begin{array}{cc} G \vdash_{(\mathrm{f},\mathrm{f})} \ell * \ell' : ((S, \Gamma), N, (T, \Delta)) \rightarrow p & \triangleright G \\ G \vdash_{(\mathrm{f},\mathrm{f})} \ell' * \ell'' : ((T, \Delta), N, (R, \Sigma)) \rightarrow p & dom\ T \cup cod\ \Delta \subseteq dom\ N \end{array}}{G \vdash_{(\mathrm{f},\mathrm{f})} \ell * \ell'' : ((S, \Gamma), N, (R, \Sigma)) \rightarrow p}$$

Intermediate Summary

- Presented basic derivation systems for unstructured bytecode
- Low assignments under high branches admitted
- compatibility with and extensibility by transformation rules (implementation interpretes labels ℓ and ℓ' w.r.t. possibly different programs/methods)
- Soundness w.r.t. termination-insensitive non-interference
- Co-termination $\rightsquigarrow$ termination-sensitive non-interference

Can this be extended to objects and methods?

Current work: Objects and methods

- Instructions: getf $C.f$, putf $C.f$, new C, invStat $C.M$
- Extension of states by heap component:

$$\begin{array}{rcl} l \in \mathbb{L} & & \\ v \in \mathcal{V} & ::= & \textbf{int } i \mid \textbf{loc } a \mid \textbf{null} \\ h \in \mathit{Heap} & ::= & \mathbb{L} \rightharpoonup_{\mathit{fin}} \mathit{Obj} \\ \mathit{Obj} & = & \mathit{Classnames} \times (\mathcal{F} \rightharpoonup_{\mathit{fin}} \mathcal{V}) \\ \mathit{States} \ni s & = & (\mathcal{O}, \sigma, h) \end{array}$$

- Operational rules, e.g.:

$$\frac{M(\ell) = (\text{getf } C.f) \quad h{\downarrow}a = (C, F) \quad F{\downarrow}f = v}{\ell, (\textbf{loc } a :: \mathcal{O}, \sigma, h) \rightarrow \ell + 1, (v :: \mathcal{O}, \sigma, h)}$$

$$\frac{\begin{array}{c} M(\ell) = \text{invStat } C.m \quad \mathcal{P}@(C.m) = (\mathit{params}, \mathit{code}, \ell_0) \\ (\mathit{params}, \mathit{code}, \ell_0), \ell_0, ([], \tau, h) \Downarrow h', v \quad \mathit{Frame}(\mathcal{O}, \mathit{params}, \tau, \mathcal{O}') \end{array}}{\ell, (\mathcal{O}, \sigma, h) \rightarrow \ell + 1, (v :: \mathcal{O}', \sigma, h')}$$

Heap abstractions & rule format

- Abstract Objects: $\mathcal{OBJ} \ni obj = (C, F)$ where $C : Classnames$ and $F : \mathcal{F} \rightharpoonup_{fin} \mathcal{C}$
- Abstract Heap: $\mathcal{H} = \mathcal{C} \rightharpoonup_{fin} \mathcal{OBJ}$
- Abstract states have shape (S, Γ, H)
- Administrative maps also store types: $N \in \mathcal{C} \rightharpoonup_{fin} Tp \times \mathcal{L}$
- Method invocation necessitates introduction of post-RSD's
- Rule format: $G \vdash_b M, \ell * M', \ell' : q, \beta \to p, \beta'$

$$\textsc{GetF}\ \frac{M(\ell) = \text{getf } C.f \quad H{\downarrow}\gamma = (D, F) \quad F{\downarrow}f = \delta \qquad G \vdash_{\mathrm{t}} M, (\ell+1) * M', \ell' : q, ((\delta :: S, \Gamma, H), N, (T, \Delta, K)) \to p, \beta}{G \vdash_{\mathrm{f}} M, \ell * M', \ell' : q, ((\gamma :: S, \Gamma, H), N, (T, \Delta, K)) \to p, \beta}$$

Heap abstractions & rule format

- Abstract Objects: $\mathcal{OBJ} \ni obj = (C, F)$ where $C : Classnames$ and $F : \mathcal{F} \rightharpoonup_{fin} \mathcal{C}$
- Abstract Heap: $\mathcal{H} = \mathcal{C} \rightharpoonup_{fin} \mathcal{OBJ}$
- Abstract states have shape (S, Γ, H)
- Administrative maps also store types: $N \in \mathcal{C} \rightharpoonup_{fin} Tp \times \mathcal{L}$
- Method invocation necessitates introduction of post-RSD's
- Rule format: $G \vdash_b M, \ell * M', \ell' : q, \beta \rightarrow p, \beta'$

$$\textsc{GetF}\ \frac{\begin{array}{c} M(\ell) = \text{getf } C.f \quad H{\downarrow}\gamma = (D, F) \quad F{\downarrow}f = \delta \\ G \vdash_{\mathrm{t}} M, (\ell + 1) * M', \ell' : q, ((\delta :: S, \Gamma, H), N, (T, \Delta, K)) \rightarrow p, \beta \end{array}}{G \vdash_{\mathrm{f}} M, \ell * M', \ell' : q, ((\gamma :: S, \Gamma, H), N, (T, \Delta, K)) \rightarrow p, \beta}$$

Allocation rules

$$\textsc{New}\ \frac{\begin{array}{c} M(\ell) = \mathsf{new}\ C \quad \beta = ((S, \Gamma, H), N, (T, \Delta, K)) \quad \gamma \notin domN \\ \beta'' = ((\gamma :: S, \Gamma, H[\gamma \mapsto (C, [])]), N[\gamma \mapsto (C, qq)], (T, \Delta, K)) \\ G \vdash_{\mathrm{t}} M, \ell + 1 * M', \ell' : q, \beta'' \rightarrow p, \beta' \end{array}}{G \vdash_{\mathrm{f}} M, l * M', \ell' : q, \beta \rightarrow p, \beta'}$$

$$\textsc{NewNew}\ \frac{\begin{array}{c} M(\ell) = \mathsf{new}\ C \quad M'(\ell') = \mathsf{new}\ C \\ \beta = ((S, \Gamma, H), N, (T, \Delta, K)) \qquad \gamma \notin dom\ N \\ \beta'' = ((\gamma :: S, \Gamma, H[\gamma \mapsto (C, [])]), N[\gamma \mapsto (C, qq)], (\gamma :: T, \Delta, K[\gamma \mapsto (C, [])])) \\ G \vdash_{\mathrm{t}} M, \ell + 1 * M', \ell' + 1 : q, \beta'' \rightarrow p, \beta' \end{array}}{G \vdash_{\mathrm{f}} M, \ell * M', \ell' : q, \beta \rightarrow p, \beta'}$$

Observation

Since the operational semantics also stores newly created objects at fresh locations, we know that colours γ_1, γ_2 with $N{\downarrow}\gamma_i = (C_i, q_i)$ may be interpreted as different addresses.

Local reasoning: frame rule

Motivation: method invocation rule that allows us to reason only about the locally relevant part of the heap

Separation into two rules: invocation (here: two-sided) ...

$$\frac{\begin{array}{c} M(\ell) = \mathrm{invStat}\ C.m \qquad M'(\ell') = \mathrm{invStat}\ C'.m' \\ \mathcal{P}@(C.m) = M_1 \qquad \mathcal{P}@(C'.m') = M_1' \\ M_1 = (pars_0, code_0, \ell_0) \qquad M_1' = (pars_1, code_1, \ell_1) \\ \beta = ((S_1, \Gamma, H), N, (T_1, \Delta, K)) \qquad q' \sqsubseteq q \\ (S_1, pars_0, \Gamma_1, S) : absFrame \qquad (T_1, pars_1, \Delta_1, T) : absFrame \\ G \vdash_{\mathrm{t}} M, \ell + 1 * M', \ell' + 1 : q', ((\gamma :: S, \Gamma, H_1), N_1, (\gamma' :: T, \Delta, K_1)) \rightarrow p, \delta \\ \beta_1 = (([], \Gamma_1, H), N, ([], \Delta_1, K)) \\ G \vdash_{\mathrm{t}} M_1, \ell_0 * M_1', \ell_1 : q, \beta_1 \rightarrow q', (([\gamma], [\,], H_1), N_1, ([\gamma'], [\,], K_1)) \end{array}}{G \vdash_{\mathrm{f}} M, \ell * M', \ell' : q, \beta \rightarrow p, \delta}$$

... plus

$$\textsc{Frame}\ \frac{\begin{array}{c} G \vdash_b M, \ell * M', \ell' : q, \beta \rightarrow p, \delta \\ \beta \oplus (H, N, K) = \beta_1 \qquad \delta \oplus (H, N, K) = \delta_1 \end{array}}{G \vdash_b M, \ell * M', \ell' : q, \beta_1 \rightarrow p, \delta_1}$$

Local reasoning: frame rule

Motivation: method invocation rule that allows us to reason only about the locally relevant part of the heap

Separation into two rules: invocation (here: two-sided) ...

$$\frac{\begin{array}{c} M(\ell) = \mathsf{invStat}\ C.m \qquad M'(\ell') = \mathsf{invStat}\ C'.m' \\ \mathcal{P}@(C.m) = M_1 \qquad \mathcal{P}@(C'.m') = M_1' \\ M_1 = (pars_0, code_0, \ell_0) \qquad M_1' = (pars_1, code_1, \ell_1) \\ \beta = ((S_1, \Gamma, H), N, (T_1, \Delta, K)) \qquad q' \sqsubseteq q \\ (S_1, pars_0, \Gamma_1, S) : absFrame \qquad (T_1, pars_1, \Delta_1, T) : absFrame \\ G \vdash_{\mathrm{t}} M, \ell + 1 * M', \ell' + 1 : q', ((\gamma :: S, \Gamma, H_1), N_1, (\gamma' :: T, \Delta, K_1)) \to p, \delta \\ \beta_1 = (([], \Gamma_1, H), N, ([], \Delta_1, K)) \\ G \vdash_{\mathrm{t}} M_1, \ell_0 * M_1', \ell_1 : q, \beta_1 \to q', (([\gamma], [\,], H_1), N_1, ([\gamma'], [\,], K_1)) \end{array}}{G \vdash_{\mathrm{f}} M, \ell * M', \ell' : q, \beta \to p, \delta}$$

... plus

$$\textsc{Frame}\ \frac{\begin{array}{c} G \vdash_b M, \ell * M', \ell' : q, \beta \to p, \delta \\ \beta \oplus (H, N, K) = \beta_1 \qquad \delta \oplus (H, N, K) = \delta_1 \end{array}}{G \vdash_b M, \ell * M', \ell' : q, \beta_1 \to p, \delta_1}$$

Current approach

- Well-definedness of RSD's includes (some) type-correctness conditions
- Concrete heaps are required to injectively contain at least addresses for all abstract locations (and type-correct objects at these locations), but may contain additional objects
- Interpretation of judgements includes frame condition, by universally quantifying over all (separated) heap extensions

Status:

- (Multi-level) one-sided and two-sided rules derived for
  - new, getfield, putfield,
  - invokeStatic, invokeVirtual
- (Two-sided) frame rule

Final slide: current & future work

Current & future work

Current work: subtyping, i.e. refinement on RSD's with heaps

- Refinement of administrative stucture N: additional entries? subclassing of existing entries?
- Refinement of abstract heaps: additional abstract objects? Sublassing of existing entries? Additional abstract fields of existing objects?
- Interaction of these choices with frame condition

Future work

- Transformation rules involving objects
- Co-termination and termination-sensitive non-interference for objects (will require side-condition in method rules)
- Encoding of other type systems for NI and/or transformation frameworks (translation validation, Tarmo/Ando)