10 Imperative Featherweight Java

10.1 Sintassi

IFJ è un imperative core calculus per Java, è tratto da
G. Bierman, M.J. Parkinson, A.M. Pitts "MJ: An imperative core calculus for Java and Java with effects", Technical Report 563, University of Cambridge - Computer Laboratory, 2003
e da A. Ahern, N. Yoshida "Formalising Java RMI with explicit code mobility", OOPSLA 2005.

Per semplificare il linguaggio, non ci sono condizionali, loop e tipi base. La sintassi sottolineata occorre solo runtime e non può dunque essere usata dal programmatore nel programma sorgente.

Classi	CL	::=	$\mathsf{class}\, C\, \mathsf{extends}\, D\, \{\tilde{A}\, f;\ K\ \tilde{M}\}$
Metodi	M	::=	$C\ m\ (\tilde{C} x)\{\mathsf{return}\, e\}$
Costruttori	K	::=	$C(\tilde{A}\, g, \tilde{B}\, f)\{\mathsf{super}(\tilde{g});\ \mathsf{this}.\tilde{f} = \tilde{f}\}$
Espressioni	e	::=	$v \mid x \mid e.f \mid e; e \mid C\ x = e\,;\ e \mid x = e$
		|	$e.f = e \mid \mathsf{new}\, C(\tilde{e}) \mid e.m(e) \mid \underline{\mathsf{NPE}}$
Valori	v	::=	$\mathsf{null} \mid \underline{o}$
Store	σ	::=	$\underline{\varnothing \mid \sigma \cdot [x \mapsto v] \mid \sigma \cdot [o \mapsto (C, \tilde{f} : v)]}$
Configurations	F	::=	$\underline{\langle \sigma\ ,\ e \rangle}$

Le espressioni contengono anche le tipiche espressioni imperative: la dichiarazione di variabili, l'assegnamento di variabili e campi dato, e la sequenza. C'è inoltre l'espressione NPE, che sta per una `NullPointerException`, che occorre solo dinamicamente. In questo calcolo imperativo i valori sono null e o, che denota un object identifier, un riferimento ad un oggetto, istanza di una classe. Una configurazione rappresenta uno stato della macchina virtuale: è costituita dall'espressione corrente e e da uno store σ, una mappa che associa un valore ad ogni variabile e associa un oggetto ad ogni riferimento. Un oggetto in memoria è rappresentato dalla classe (il tipo) dell'oggetto e dal valore dei suoi campi dato (osserva che non ci sono i metodi). Indichiamo con $\sigma[x \mapsto v]$ lo store σ in cui ad x si associa il nuovo valore v e indichiamo con $\sigma[o.f \mapsto v]$ lo store σ in cui la entry $\sigma(o)$ modifica il valore del campo dati f in v, se un tale campo dati esiste, altrimenti $\sigma[o.f \mapsto v]$ è indefinito .

Anche in IFJ assumiamo che un programma contenga una Class Table CT (ben fatta), che associa ad ogni nome di classe C una dichiarazione CL.

EXAMPLE 10.1. La memoria può contenere alias, ad esempio: $\sigma = [o \mapsto \mathsf{Object}] \cdot [x \mapsto o] \cdot [y \mapsto o]$, che è molto diverso dallo store $\sigma' = [o_1 \mapsto \mathsf{Object}] \cdot [x \mapsto o_1] \cdot [o_2 \mapsto \mathsf{Object}] \cdot [y \mapsto o_2]$. Si osservi invece che le memorie $\sigma_1[o \mapsto \mathsf{null}]$ e $\sigma'_1 = [x \mapsto \mathsf{Object}]$ non sono ben definite. □

10.1.1 Semantica Operazionale

La semantica operazionale è definta come una relazione di transizione tra configurazioni: $\langle \sigma\ ,\ e \rangle \longrightarrow \langle \sigma'\ ,\ e' \rangle$.

(NEW)
$$\frac{o \notin Dom(\sigma) \quad fields(C) = \tilde{A}\, f \quad |\tilde{f}| = |\tilde{v}|}{\langle \sigma\ ,\ \mathsf{new}\, C(\tilde{v}) \rangle \longrightarrow \langle \sigma \cdot [o \mapsto (C, \tilde{f} : v)]\ ,\ o \rangle}$$

(DICHIAR)
$$\frac{x \notin Dom(\sigma)}{\langle \sigma\ ,\ C\ x = v; e \rangle \longrightarrow \langle \sigma \cdot [x \mapsto v]\ ,\ e \rangle}$$

(VAR)
$$\frac{}{\langle \sigma\ ,\ x \rangle \longrightarrow \langle \sigma\ ,\ \sigma(x) \rangle}$$

(ASSEGN)
$$\frac{}{\langle \sigma\ ,\ x = v \rangle \longrightarrow \langle \sigma[x \mapsto v]\ ,\ v \rangle}$$

Si osservi che la dichiarazione di variabile evolve solamente se è seguita da ulteriori espressioni, non può cioè essere l'ultima istruzione.

$$\text{(ASSEGN CAMPI)}\quad \frac{\sigma(o) = (C, f \tilde{:} v) \quad f \in \tilde{f}}{\langle \sigma\,,\, o.f = v' \rangle \longrightarrow \langle \sigma[o.f \mapsto v']\,,\, v' \rangle} \qquad \text{(FIELD)}\quad \frac{\sigma(o) = (C, f \tilde{:} v) \quad f_i \in \tilde{f}}{\langle \sigma\,,\, o.f_i \rangle \longrightarrow \langle \sigma\,,\, v_i \rangle}$$

$$\text{(METHOD CALL)}\quad \frac{\sigma(o) = (C, ...) \quad mbody(m, C) = (\tilde{x}, e) \quad \tilde{x} \notin Dom(\sigma) \quad |\tilde{v}| = |\tilde{x}|}{\langle \sigma\,,\, o.m(\tilde{v}) \rangle \longrightarrow \langle \sigma \cdot [\tilde{x} \mapsto \tilde{v}]\,,\, e\{\mathsf{this} := o\} \rangle}$$

Si osservi che la regola (METHOD CALL) sporca lo store con tutte le variabili corrispondenti ai parametri formali, che una volta terminato il metodo potrebbero essere eliminati dallo store. Per gestire uno scope lessicale si potrebbe aggiungere la seguente regola:

$$\text{(GARBAGE COLLECTION)}\quad \frac{x \notin \mathrm{fv}(e)}{\langle \sigma \cdot [x \mapsto v]\,,\, e \rangle \longrightarrow \langle \sigma\,,\, e \rangle}$$

Introdurre questa regola farebbe però perdere il determinismo della relazione di transizione. D'altra parte scrivere una strategia di riduzione che applica la garbage collection appena termina l'esecuzione di un metodo, oppure appena una variabile non compare più nel termine che evole non è cosa facile (Si veda l'articolo sul linguaggio MJ per una gestione completa degli scope). Per semplicità assumiamo dunque un'infinità di nomi distinti di variabili, che non vengono mai cancellati dallo store.

EXERCISE 10.2. Scrivere una regola analoga per eliminare i riferimenti non più usati. □

Si osservi che per come sono definti gli store, dato uno store σ ed un object identifier o, allora $\sigma(o) = (C, ...)$ oppure $o \notin Dom(\sigma)$. Nel secondo caso termini come $o.f, o.m()$ oppure $o.f = v$ sono termini stuck. Diverso è invece il caso delle seguenti regole, dove si tenta di dereferenziare un riferimento nullo:

$$\text{(ERR-NULL 1)}\quad \frac{}{\langle \sigma\,,\, \mathsf{null}.f \rangle \longrightarrow \langle \sigma\,,\, \mathsf{NPE} \rangle} \qquad \text{(ERR-NULL 2)}\quad \frac{}{\langle \sigma\,,\, \mathsf{null}.f = v \rangle \longrightarrow \langle \sigma\,,\, \mathsf{NPE} \rangle} \qquad \text{(ERR-NULL 3)}\quad \frac{}{\langle \sigma\,,\, \mathsf{null}.m(\tilde{v}) \rangle \longrightarrow \langle \sigma\,,\, \mathsf{NPE} \rangle}$$

$$\text{(SEQ)}\quad \frac{}{\langle \sigma\,,\, v; e \rangle \longrightarrow \langle \sigma\,,\, e \rangle} \qquad \text{(CONG)}\quad \frac{\langle \sigma\,,\, e \rangle \longrightarrow \langle \sigma'\,,\, e' \rangle}{\langle \sigma\,,\, E[e] \rangle \longrightarrow \langle \sigma'\,,\, E[e'] \rangle} \qquad \text{(ERR)}\quad \frac{}{\langle \sigma\,,\, E[\mathsf{NPE}] \rangle \longrightarrow \langle \sigma\,,\, \mathsf{NPE} \rangle}$$

Dove gli evaluation contexts sono i seguenti:

$$\begin{aligned} E \;::=\;& [\,] \mid E.f \mid E; e \mid x = E \mid E.f = e \mid \\ & o.f = E \mid \mathsf{new}\, C(\tilde{v}, E, \tilde{e}) \mid E.m(\tilde{e}) \mid o.m(\tilde{v}, E, \tilde{e}) \mid C\; x = E \end{aligned}$$

Ad esempio, se $\langle \sigma\,,\, e_1 \rangle \longrightarrow \langle \sigma'\,,\, e_1' \rangle$ allora per la regola (CONG) si ha $\langle \sigma\,,\, \mathsf{new}\, C(v_1, e_1, e_2).m(e_3) \rangle \longrightarrow \langle \sigma'\,,\, \mathsf{new}\, C(v_1, e_1', e_2).m(e_3) \rangle$ con $E[\,] = \mathsf{new}\, C(v_1, [\,], e_2).m(e_3)$. Come ulteriore esempio, sia $e = \mathsf{new}\, C(v_1, o.m(e_1), e_2)$, allora in questo caso si appplica la regola (CONG) con $E[\,] = E_1[\,E_2[\,]\,]$ dove $E_1[\,] = \mathsf{new}\, C(v_1, [\,], e_2)$ ed $E_2[\,] = o.m([\,])$.

EXERCISE 10.3. Descrivere il comportamento del seguente programma:

```
class D extends Object {
        Object f;
        D(Object f) { super(); this.f=f;}
        Object m() { return this; }
}
```

```
class C extends D {
        C(Object f) { super(f); }
        Object m() {return this;}
}
Object z=new Object();
C x=new C(z);
C y=new D(x);
x.m();  x=y; x.m()
y.f=new Object();
z=null; x.f=z; y.f.m();
```

□

10.1.2 Sistema di Tipi

Anche in IFJ i tipi sono i nomi delle classi. Tra le assunzioni di tipo aggiungiamo invece anche le assunzioni di tipo per i riferimenti agli oggetti:

$$\textit{Type environment}\ \Gamma \quad ::= \quad \varnothing \mid \Gamma, x : C \mid \Gamma, o : C$$

Le regole di subtyping e di buona formazione di metodi e classi sono identiche a FJ.

Subtyping

$$\frac{}{C <: C}\ (\textsc{Rifl}) \qquad \frac{C <: D \quad D <: E}{C <: E}\ (\textsc{Trans}) \qquad \frac{CT(C) = \mathsf{class}\ C\ \mathsf{extends}\ D\ \{...\}}{C <: D}\ (\textsc{Class})$$

Well Formedness

(M OK IN C)

$$\frac{\tilde{x} : \tilde{C}, \mathsf{this} : C \vdash t : B \quad B <: A \quad CT(C) = \mathsf{class}\ C\ \mathsf{extends}\ D\ \{...\} \quad override(m, D, \tilde{C} \to A)}{A\ m\ (\tilde{C\,x})\ \{\mathsf{return}\ t;\}\ OK\ in\ C}$$

(C OK))

$$\frac{K = C(\tilde{D\,g}, \tilde{C\,f})\ \{\mathsf{super}(\tilde{g});\ \mathsf{this}.\tilde{f} = \tilde{f};\} \quad fields(D) = \tilde{D\,g} \quad \tilde{M}\ OK\ in\ C}{\mathsf{class}\ C\ \mathsf{extends}\ D\ \{\tilde{C\,f};\ K\ \ \tilde{M}\}\ OK}$$

Tipi delle configurazioni

$$\frac{\Gamma \vdash e : C \quad \Gamma \vdash \sigma}{\Gamma \vdash \langle \sigma\ ,\ e \rangle}\ (\textsc{Conf})$$

type store

$$\frac{}{\Gamma \vdash \varnothing}\ (\textsc{Empty Store}) \qquad \frac{\Gamma \vdash \sigma \quad x \notin Dom(\sigma) \quad \Gamma \vdash x : C \quad \Gamma \vdash v : D \quad D <: C}{\Gamma \vdash \sigma \cdot [x \mapsto v]}\ (\textsc{Var Store})$$

(OID STORE)

$$\frac{\Gamma \vdash \sigma \quad o \notin Dom(\sigma) \quad \Gamma \vdash o : C \quad fields(C) = \tilde{A\,f} \quad \Gamma \vdash \tilde{v} : \tilde{B} \quad \tilde{B} <: \tilde{A}}{\Gamma \vdash \sigma \cdot [o \mapsto (C, \tilde{f : v})]}$$

Tipi delle espressioni

$$\frac{}{\Gamma \vdash \mathsf{null} : C}\ (\textsc{Null}) \qquad \frac{o : C \in \Gamma}{\Gamma \vdash o : C}\ (\textsc{Oid}) \qquad \frac{x : C \in \Gamma}{\Gamma \vdash x : C}\ (\textsc{Var})$$

$$\frac{\Gamma \vdash e : C \quad field(C) = \tilde{D f}}{\Gamma \vdash e.f_i : D_i}\ (\textsc{Field}) \qquad \frac{\Gamma \vdash e_1 : C \quad \Gamma \vdash e_2 : D}{\Gamma \vdash e_1; e_2 : D}\ (\textsc{Seq}) \qquad \frac{\Gamma \vdash e : C \quad C <: D \quad \Gamma \vdash x : D}{\Gamma \vdash x = e : C}\ (\textsc{Assegn})$$

$$\frac{\Gamma \vdash e.f : D \quad C <: D \quad \Gamma \vdash e' : C}{\Gamma \vdash e.f = e' : C}\ (\textsc{Field Assign}) \qquad \frac{\Gamma \vdash e : C \quad C <: D \quad \Gamma, x : D \vdash e' : C'}{\Gamma \vdash D\ x = e; e' : C'}\ (\textsc{Dichiar})$$

$$\frac{fileds(C) = \tilde{D f} \quad \Gamma \vdash \tilde{e} : \tilde{C} \quad \tilde{C} <: \tilde{D} \quad |\tilde{e}| = |\tilde{f}|}{\Gamma \vdash \mathsf{new}\, C(\tilde{e}) : C}\ (\textsc{New})$$

$$\frac{\Gamma \vdash e : C \quad mtype(m, C) = \tilde{A} \to B \quad \Gamma \vdash \tilde{e} : \tilde{D} \quad \tilde{D} <: \tilde{A} \quad |\tilde{e}| = |\tilde{A}|}{\Gamma \vdash e.m(\tilde{e}) : B}\ (\textsc{Method})$$

EXERCISE 10.4. Dare una derivazione di tipo per il termine C x=o$_1$; o$_1$.f=y; x=o$_2$ □

Si osservi che non c'è una regola di tipo per il termine NPE, quindi ogni configurazione della forma $\langle \sigma\ ,\ E[\,\mathsf{NPE}\,]\rangle$ non sarà ben tipata. Nel mini linguaggio funzionale avevamo invece dato una regola di tipo per tipare anche le espressioni throw v, ma ciò era perché il linguaggio permetteva non solo di sollevare, ma anche di gestire tali eccezioni tramite un try-catch, cosa invece non possibile in IFJ per NPE. Di conseguenza il teorema di subject reduction avrà un enunciato diverso.

Lemma 10.5 (Substitution)**.** *Se* $\Gamma, \mathsf{this} : C \vdash e : D$ *e* $\Gamma \vdash o : C'$ *con* $C' <: C$ *allora* $\Gamma \vdash e\{\mathsf{this} := o\} : D'$ *per qualche* $D' <: D$. □

Theorem 10.6 (Subject Reduction)**.**

1. *Se* $\Gamma \vdash e : C$ *e* $\Gamma \vdash \sigma$ *ed inoltre* $\langle \sigma\ ,\ e\rangle \longrightarrow \langle \sigma'\ ,\ e'\rangle$ *con* $e' \neq E[\,\mathsf{NPE}\,]$*, allora* $\Gamma \vdash e' : C'$ *con* $C' <: D$ *e* $\Gamma \vdash \sigma'$
2. *Se* $\Gamma \vdash \langle \sigma\,, e\rangle$ *e* $\langle \sigma\,, e\rangle \longrightarrow \langle \sigma'\,, e'\rangle$ *con* $e' \neq E[\,\mathsf{NPE}\,]$*, allora* $\Gamma \vdash \langle \sigma'\,, e'\rangle$.

□

Diciamo che una configurazione $\langle \sigma\ ,\ e\rangle$ è chiusa quando la memoria associa un valore ad ogni variabile libera dell'esspressione e e ogni object identifier che compare in e è definito in σ, cioè quando $\mathrm{fv}(e) \cup \mathsf{fref}(e) \subseteq (\sigma)$.

Theorem 10.7 (Progress)**.** *Se* $\langle \sigma\ ,\ e\rangle$ *è chiusa e ben tipata, allora* $e = v$ *oppure* $\exists \sigma', e'.\ \langle \sigma\ ,\ e\rangle \longrightarrow \langle \sigma'\ ,\ e'\rangle$. □

Theorem 10.8 (Safety)**.** *Se* $\varnothing \vdash \langle \emptyset\,, e\rangle$*, allora* $\exists \sigma, v.\ \langle \emptyset\,, e\rangle \longrightarrow^* \langle \sigma\,, v\rangle$ *oppure* $\langle \emptyset\,, e\rangle \longrightarrow^* \langle \sigma\,, \mathsf{NPE}\rangle$. □