FJ: sintassi

Indichiamo con $A, B, C, D...$ i nomi di classi $\tilde{t} = t_1..t_n$, $\widetilde{A\,f} = A_1\,f_1...A_n\,f_n$

Dic classe	CL	::=	class C extends $D\,\{\widetilde{A\,f};\ K\ \tilde{M}\}$
Dic costruttore	K	::=	$C(\widetilde{A\,g}, \widetilde{B\,f})\,\{\text{super}(\tilde{g});\ \text{this}.\tilde{f} = \tilde{f}\}$
Dic metodo	M	::=	$C\ m(\widetilde{A\,x})\{\text{return } t;\}$

Termini t ::= $x \mid t.f \mid t.m(\tilde{t}) \mid \text{new } C(\tilde{t}) \mid (C)\,t$

Valori v ::= $\text{new } C(\tilde{v})$

Programma ::= (CT, t). Assumiamo Class Table CT ben definita.

FJ: subtyping

$$\frac{CT(C) = \text{class } C \text{ extends } D\{...\}}{C <: D}$$

$$\frac{}{C <: C}$$

$$\frac{C <: D \qquad D <: E}{C <: E}$$

Una class table ben definita induce subtyping senza cicli

FJ: Semantica Operazionale

$$\frac{\mathit{fields}(C) = \widetilde{C\,f} \quad f_i \in \tilde{f}}{(\text{new } C(\tilde{v})).f_i \longrightarrow v_i}$$

Lookup di campi: definto da:

$$\mathit{fields}(\text{Object}) = \emptyset$$

$$\frac{CT(C) = \text{class } C \text{ extends } D\,\{\widetilde{C\,f};\ K\ \tilde{M}\} \qquad \mathit{fields}(D) = \widetilde{D\,g}}{\mathit{fields}(C) = \widetilde{D\,g},\ \widetilde{C\,f}}$$

FJ: Semantica Operazionale

$$\frac{\mathit{mbody}(m, C) = (\tilde{x}, t)}{(\text{new } C(\tilde{v})).m(\tilde{v}') \longrightarrow t\{\tilde{x} := \tilde{v}',\ \text{this} := \text{new } C(\tilde{v})\}}$$

Lookup di definizioni di metodi definto da:

$$\frac{CT(C) = \text{class } C \text{ extends } D\,\{\widetilde{C\,f};\ K\ \tilde{M}\} \qquad B\ m\ (\widetilde{B\,x})\ \{\text{return } t;\} \in \tilde{M}}{\mathit{mbody}(m, C) = (\tilde{x}, t)}$$

$$\frac{CT(C) = \text{class } C \text{ extends } D\,\{\widetilde{C\,f};\ K\ \tilde{M}\} \qquad m \text{ non definito in } \tilde{M}}{\mathit{mbody}(m, C) = \mathit{mbody}(m, D)}$$

FJ: Semantica Operazionale

$$\frac{t \longrightarrow t'}{t.f \longrightarrow t'.f} \qquad \frac{t \longrightarrow t'}{t.m(\tilde{t}) \longrightarrow t'.m(\tilde{t})}$$

$$\frac{t_i \longrightarrow t_i'}{\text{new } C(\tilde{v}, t_i, \tilde{t}) \longrightarrow \text{new } C(\tilde{v}, t_i', \tilde{t})}$$

$$\frac{t_i \longrightarrow t_i'}{v.m(\tilde{v}, t_i, \tilde{t}) \longrightarrow v.m(\tilde{v}, t_i', \tilde{t})}$$

FJ: Semantica Operazionale (Cast)

$$\frac{C <: D}{(D)(\text{new } C(\tilde{v})) \longrightarrow \text{new } C(\tilde{v})}$$

$$\frac{t \longrightarrow t'}{(C)t \longrightarrow (C)t'}$$

Il cast ha successo solo alzando il tipo di un oggetto, altrimenti il termine è **stuck**. In FJ non ci sono eccezioni.

Es. sia $C <: B <: A$

(B) ((A) new C()) $\longrightarrow$ (B) new C() $\longrightarrow$ new C()

FJ: type system

Il type system utilizza tre forme di giudizi:

- $\Gamma \vdash t : C$
  per indicare che il termine t ha tipo C in Γ.
- $D\ m\ (\tilde{C}\,x)\ \{\text{return } t;\}\ OK\ in\ C$
  per indicare che il metodo m è ben formato se occorre nella classe C
- class C extends $D\,\{\tilde{C}\,f;\ K\ \tilde{M}\}\ OK$
  per indicare che la definizione della classe C è ben formata.

Il programma (CT, t) è **ben tipato** se sono derivabili sia $CT\ OK$ che $\emptyset \vdash t : C$ per qualche C.

FJ: type system

$$\frac{x : C \in \Gamma}{\Gamma \vdash x : C}$$

$$\frac{\Gamma \vdash t : C \quad fields(C) = \widetilde{D\,f} \quad f_i \in \tilde{f}}{\Gamma \vdash t.f_i : D_i}$$

$$\frac{fields(C) = \widetilde{D\,f} \quad \Gamma \vdash \tilde{t} : \tilde{A} \quad \tilde{A} <: \tilde{D}}{\Gamma \vdash \text{new } C(\tilde{t}) : C}$$

FJ: type ststem

$$\frac{\Gamma \vdash t' : C \quad mtype(m, C) = \tilde{A} \to B \quad \Gamma \vdash \tilde{t} : \tilde{D} \quad \tilde{D} <: \tilde{A}}{\Gamma \vdash t'.m(\tilde{t}) : B}$$

Dove il predicato ausiliario *mytype* è definito come segue:

$$\frac{CT(C) = \text{class } C \text{ extends } D\,\{\tilde{C}\,f;\ K\ \tilde{M}\} \qquad B\ m\ (\tilde{A}\,x)\ \{\text{return } t;\} \in M}{mtype(m, C) = \tilde{A} \to B}$$

$$\frac{CT(C) = \text{class } C \text{ extends } D\,\{\tilde{C}\,f;\ K\ \tilde{M}\} \qquad m \text{ non definito in } M}{mtype(m, C) = mtype(m, D)}$$

FJ: type system (Cast)

(Up Cast)

$$\frac{\Gamma \vdash t : D \quad D <: C}{\Gamma \vdash (C)t : C}$$

(Down Cast)

$$\frac{\Gamma \vdash t : D \quad C <: D \quad C \neq D}{\Gamma \vdash (C)t : C}$$

(B) ((A) new C()) $\longrightarrow$ (B) new C() $\longrightarrow$ new C() con $C <: B <: A$

$\emptyset \vdash$ (A) new C() : A tipo statico A tipo dinamico C

$\emptyset \vdash$ (B)((A) new C()) : B tipo statico B tipo dinamico C

FJ: type system (Cast)

(Up Cast)

$$\frac{\Gamma \vdash t : D \quad D <: C}{\Gamma \vdash (C)t : C}$$

(Down Cast)

$$\frac{\Gamma \vdash t : D \quad C <: D \quad C \neq D}{\Gamma \vdash (C)t : C}$$

(A) ((Object) new $B()$) $\longrightarrow$ (A) new $B()$ A e B non in relazione

$\emptyset \vdash$ (A)((Object) new B()) : A ma $\emptyset \not\vdash$ (A) new B() : **?!**

Non vale il teorema di Preservazione dei tipi!

FJ: type system (Cast)

(Up Cast)

$$\frac{\Gamma \vdash t : D \quad D <: C}{\Gamma \vdash (C)t : C}$$

(Down Cast)

$$\frac{\Gamma \vdash t : D \quad C <: D \quad C \neq D}{\Gamma \vdash (C)t : C}$$

$$\frac{\Gamma \vdash t : D \quad C \not<: D \quad D \not<: C \quad warning}{\Gamma \vdash (C)t : C}\ (StupidCast)$$

Un programma senza 'stupid cast' può contenere runtime uno 'stupid cast':

(A) ((Object) new $B()$) $\longrightarrow$ (A) new $B()$ A e B non in relazione

$\emptyset \vdash$ (A)((Object) new B()) : A ma $\emptyset \not\vdash$ (A) new B() : **A**

Il tipo si preserva... ma

Un programma ben tipato evolve in un termine stuck (di tipo 3.)!

FJ: type system (Class Table)

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

$$\frac{\tilde{x} : \tilde{C}, \text{this} : C \vdash t : B \qquad B <: A \qquad CT(C) = \text{class}\ C\ \text{extends}\ D\,\{\ldots\} \qquad \mathit{override}(m, D, \tilde{C} \to A)}{A\ m\ (\widetilde{C\,x})\ \{\text{return}\ t;\}\ \mathit{OK\ in}\ C}$$

Dove il predicato *override* è definito nel modo seguente:

$$\frac{\text{Se}\ \ \mathit{mytype}(m, D) = \tilde{D} \to B \quad \text{allora} \quad \tilde{C} = \tilde{D}\ \text{e}\ A = B}{\mathit{override}(m, D, \tilde{C} \to A)}$$

FJ: Proprietà del sistema di tipi

Subject Reduction

Se $\Gamma \vdash t : C$ e $t \longrightarrow t'$ allora $\Gamma \vdash t' : C'$ per qualche $C' <: C$

Progress

Sia t un termine chiuso, ben tipato. Allora $t \longrightarrow t'$ oppure t è un valore, oppure $\exists E$, evaluation context, tale che $t = E[\ (C)(\text{new}\ D(\tilde{v}))\ ]$ con $D \not<: C$.

Dove gli evaluation contexts sono definiti come segue:

$$E \ ::= \ [\,] \mid E.f \mid E.m(\tilde{t}) \mid v.m(\tilde{v}, E, \tilde{t}) \mid \text{new}\ C(\tilde{v}, E, \tilde{t}) \mid (C)E$$

Es: $\text{new}\ C().m(v_1, t, t') = E[\,t\,]$ *con* $E[\,] = \text{new}\ C().m(v_1, [\,], t')$

FJ: Proprietà del sistema di tipi

Subject Reduction

Se $\Gamma \vdash t : C$ e $t \longrightarrow t'$ allora $\Gamma \vdash t' : C'$ per qualche $C' <: C$

Progress

Sia t un termine chiuso, ben tipato. Allora $t \longrightarrow t'$ oppure t è un valore, oppure $\exists E$, evaluation context, tale che $t = E[\ (C)(\text{new}\ D(\tilde{v}))\ ]$ con $D \not<: C$.

Safety

Sia t un programma ben tipato,
allora t non evolve runtime ad un termine che contiene un errore del tipo **message-not-understood**.

FJ: Proprietà del sistema di tipi

Subject Reduction

Se $\Gamma \vdash t : C$ e $t \longrightarrow t'$ allora $\Gamma \vdash t' : C'$ per qualche $C' <: C$

Progress

Sia t un termine chiuso, ben tipato. Allora $t \longrightarrow t'$ oppure t è un valore, oppure $\exists E$, evaluation context, tale che $t = E[\ (C)(\text{new}\ D(\tilde{v}))\ ]$ con $D \not<: C$.

Safety in Java

Un programma ben tipato,
non evolve runtime ad un errore **message-not-understood** ma può sollevare `ClassCastException`.

FJ: proprietà del sistema di tipi

Un termine t ben tipato è ***cast safe*** se contiene solo up-cast,
i.e. se è ben tipato senza usare le regole di tipo (DownCast) e (Stupid Cast).

Se t è cast safe e $t \longrightarrow t'$, allora anche t' è cast safe.

Safety
Sia t un programma ben tipato cast safe, allora t non evolve ad un termine stuck, i.e. se $t \longrightarrow^* t' \not\longrightarrow$ allora t' è un valore.