

Record vs oggetti: I record non hanno uno stato (non hanno accesso agli altri campi). Sono solo valori accessibili tramite il nome del campo

Chiusura where esprimibile di un'estrazione funzionale,
Combine una funzione con le variabili libere presenti nell'ambiente secondo le regole di scoping

Progresso una espressione ben tipata non si blocca a run-time

Conservazione i tipi sono preservati dalle regole di esecuzione



SK:T = s sotto tipo di T

Mini Caml

Da estendere

Type exp; \rightarrow def. del tipo delle nuove funzioni ohe
eval (NECESSARIA)

let rec eval (e:exp)(s:env) : env =

match e with \rightarrow le nuove funzioni (NECESSARIA)

Type evT = \rightarrow per la definizione di nuovi Tipi

Polimorfismo: f è polimorfo se può essere applicata a
argomenti di tipo diverso

Subsumption se $S \leq T$, ogni valore di tipo S può essere
considerato di tipo T

Inferenza di tipo: data un'espressione non tipata, ricostruisce
i tipi più generale



Object-oriented programming

Caratteristiche di un oggetto

Stato: rappresentato da gruppi di attributi / proprietà / variabili
Applicato information hiding: A non conosce lo stato di B

Funzionalità: gruppi di funzioni / metodi che l'oggetto mette a disposizione di altri oggetti

Identità (nome), ciclo di vita, locazione di memoria

Comunicano scambiandosi messaggi

Concetti chiave

Incapsulamento lo stato di un oggetto non dovrebbe essere accessibile dagli altri oggetti

Abstractazione riguarda sul comportamento di un oggetto conoscere le rappresentazioni interne

Interfaccia cosa un oggetto mette a disposizione di altri

Ereditarietà come un oggetto può fare proprie le funzionalità di un altro oggetto

Pr. di sostituzione quando un ogg può essere usato al posto di un altro

Polidomorfismo come un ogg può processare altri oggetti



Approccio object-based

Oggetti trattati come record

Compi possono essere associati a funzioni

Method accedono ai campi tramite this

Flessibile si può scrivere il codice di un oggetto prima delle classi; si possono creare varianti di un oggetto

Difficile preuire il tipo di un oggetto

Mantiene liste di prototipi per ereditarietà (ogni ogg)

Approccio class-based

Classi definiscono i contenuti degli oggetti di un certo tipo

Oggetti creati dopo come istanze di una classe

Consente controlli di tipo statici - tipo legato alle classi

Richiede maggiore disciplina nella scrittura del codice

Possibile definire classi come estensione di altre: le nuove ereditano tutti i membri e può aggiungerne/ridefinirne

Cast Tipping: Interfaccia

Inoltre quei membri pubblici deve contenere
sia la classe

- Non contiene costruttori

Cosa non COME → no problemi di er. multiple



Subtyping

Un oggetto B sottoinsieme di A dovrebbe poter essere usato
dovunque si possa usare A

Structural subtyping

Presente nei linguaggi object-based

B è sottotipo di A se contiene almeno tutti i metodi pubblici presenti anche in A

Nozione di sottotipo già ristretta ai record

Più flessibile, favonisce il polimorfismo

Nominal subtyping

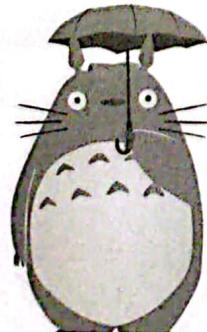
Presente nei linguaggi class-based

Il tipo di oggetto corrisponde alla classe

B è sottotipo di A se B è definito come estensione di A

«Studente extends persona»

Più semplice da verificare per l'interprete, più rigoroso per il programmatore



Structural subtyping è una relazione più debole di Nominal subtyping

LISKOV

UN oggetto del sottotipo può sostituire uno del supertipo senza influire sul comportamento dei programmi

- REGOLE

SEGNATURA: gli oggetti del sottotipo devono avere tutti i membri del supertipo + tipi compatibili nelle firme

METODI: le chiamate dei metodi del sottotipo devono comportarsi come quelle del supertipo

PROPRIETÀ: il sottotipo deve preservare tutte le proprietà del supertipo

sottoclasse deve soddisfare specifiche di superclasse

membri pubblici deve

essere

L' LISKOV

sostituire uno del supertipo
nello stesso modo

Segnatura: - gli oggetti del sottotipo devono avere tutti i membri del supertipo; - le firme dei metodi del sottotipo devono essere compatibili con quelle del supertipo;
- overriding: stessa firma, può sollevare meno eccez., può restituire un tipo più specifico

Metodi: le chiamate dei met. del sottotipo devono comportarsi come quelle del supertipo - Inabolire le precondizioni e rafforzare le postcondizioni

Proprietà: il sottotipo deve preservare tutte le proprietà degli oggetti del supertipo

Svolgere deve conservare le proprietà d'isope



Tecniche di garbage collection

Mark-Sweep

Bit di marcatura per ogni cella

Quando attivato, il programma viene sospeso

Marking

Dal root set si marcano le celle live

Sweep

Tutte le celle non marcate sono garbage e vengono restituite alla lista libera

Reset dei bit di marcatura

Opera correttamente sui cicli/strutture circolari +

Non vi è over head +

Si mette in pausa l'esecuzione -

L'heap reste frammentato -

Copying collection

Noto come Algoritmo di Cheney

Suddivisione della memoria in due parti, una sola attiva

All'attivazione, le celle live vengono copiate nella seconda porzione, se

le celle nella parte non attiva vengono restituite alla lista libera in un unico blocco



Efficiente nell'allocazione +
Evita la fragmentazione +
Duplica lo heap: bisogna di più memoria -

Generational garbage collection

"most cells that die, die young"; divisione dello heap in generazioni

Si occupano le celle live in old

Si pulisce young

Si colloca come nuovi blocchi in young

Nello practice è comune avere più (3) generazioni

Old viene pulito tramite altra tecnica di GC

Reference counting

Ogni cella ha un contatore di riferimenti

Non necessita di GC +

Over head di gestione -

Problema: strutture con cicli interni non autorizzate - memory leak

Root set, insieme dei dati effettivi (statici e su runtime stack)



Locking delle risorse condivise

metodo

coarse-grained: unico mutex per tutte le locazioni

fine-grained: mutex diverso per ogni locazione

Coarse-grained

Richiede meno lavoro al thread

Ma riduce la concorrenza

Fine-grained

Favoriscono la concorrenza

Ma portano a deadlock

Risolvere i deadlock

Prevention: regole controllate staticamente dal compilatore

Avoidance: supporto a runtime, se sta per andare in deadlock cambia l'ordine dei thread

Recovery: viene lasciato andare in deadlock e poi ripristinato a uno stato precedente



Two-phase locking (ex di d. prevention)

Ogni thread deve fare: lock in ord. crescente,
unlock in ord. decrescente

Le seguenti devono essere eseguite completamente:

Ereditarietà multiple

estendere più classi

Problemi

- Ereditare diverse implementazioni dello stesso metodo
- Diamond problem: duplicati creati da superclassi con superclassi in comune

È le soluzioni

C++ ereditarietà multiple e disambiguazione

Java interfacce per definire supertipi

Python linearizzazione delle gerarchie

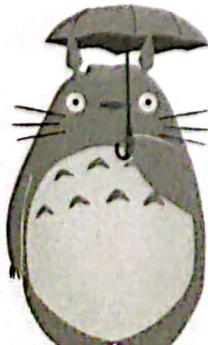
composizione di classi (mixin)

C++

Diverse implementazioni; disambiguazione nelle chiamate a metodi "c.A::foo()"

Diamond problem uso di classi virtuali; contenenti puntatori invece di copie delle superclassi (oggetto di overhead)

Implements una vera ereditarietà multiple, ma
richiede consapevolezza parte del programmatore



Java

Prevede ereditarietà singola con implementazione multiple di interfacce

Limitato ma più semplice per il programmatore

Tabella delle interfacce (itable) al posto di dispatch vector

- A seconda del tipo apparente avviene lo chiamata normale o tramite itable. Accesso a itable: compilatore scorre finché non trova tipo corrispondente (operazione costosa).
- Salvo il risultato in cache per accessi futuri.

Da Java 8 è stato introdotto l'ereditarietà multiple tramite implementazioni di default. Non è presente nessun metodo da usare in caso di ambiguità

Python

Linearizzazione delle classi tramite MRO: una lista fisica di ordinamento, deterministica e monotono.

Non necessita di una fase di compilazione.

! Esistono gerarchie non linearizzabili

Mixin

Teoricamente simile a pert. diritt.
in BD

Anziché ereditore, è una composizione.

Implementati: spesso come classi senza superclassi



Dispatching

Dispatch vector: Tabella dei metodi delle classi

Overloading: possibile definire più metodi con lo stesso nome ma firma diversa

Downcast: esplicito se è da supertipo a sottotipo. Controllo dinamico con `instanceOf`

Ricerca del metodo a tempo di esecuzione tramite `dynamic dispatch`

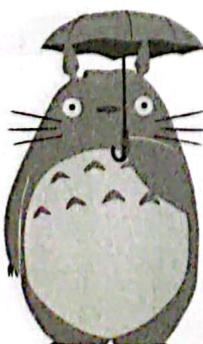
Dynamic dispatch

Parte della classe corrente e risale la gerarchia.

Esegue il primo metodo incontrato. In caso di overriding

Visita l'albero ogni volta è inefficiente. Lo SVT combina dispatch vector (Tabella di puntatori ai metodi) e sharing strutturale (le strutture delle tab delle superclassi sono riuite da quelle delle subclass, aggiungendo righe per i nuovi metodi)

Soluzione statica al dispatching



• + Generics

Consentono la definizione di classi e metodi per metri ai rispetto al tipo

In Java, il sottotipo è inferibile per le classi generiche

In Ocaml si calcola invece covarianza per i generici.

Non avviene in Java perché gli elementi sono modificabili

Ex. Creo lista dogs con cani, poi liste cani nello come

alias di dogs. A questo punto aggiungo ad emuls

cot. Pi che tipo è un elemento estratto da dogs?

Soluzione adottata: definizione lim sup delle gerarchie di tipi tramite extends o lim inf tramite super

Wildcard?

Variabile di tipo anonime, Tipo presente una volta ma non si conosce il nome

Stesse sintassi dei generics: extends per la lettura di roboti, super per la scrittura

NON si usa per lettura e scrittura

!

? vs object; object escluse string!

Array: se typed sott. di Typez, Typez è sottotipo di Typezz
programmatori pigri, presente problemi!

I generics sono totali isolati a objects (backward compatibility)



Programmazione concorrente

Shared memory

- I processi possono accedere alle stesse aree di memoria
- Si sincronizzano e comunicano leggendo variabili condivise

Message passing

- I processi accedono ad aree di memoria separate
- Si sincronizzano e comunicano usando servizi di inter-process communication

Mecanismi di sincronizzazione

Busy waiting (`Sleep(n)`): inefficiente, consuma tempo di CPU
`Sleep()` e `wakeup()`: shared memory

`send()` e `receive()`: inter-process communication

Interleaving: alternare l'esecuzione dei programmi, concorrenza. Avviene a livello macchina. È non deterministico



RUST

Concetti chiave

Variebili immutabili, ma le rende mutabili

Ownership: i dati sulla heap hanno un unico proprietario

Lifetime: durata dell'essere live di un dato

Memory Safety: controllo tipi, no pattern memory unsafe

Ownership

Proprietario unico. Quando esce della scope il valore viene eliminato

Quando il dato viene trasmesso a funzioni la proprietà viene trasferita.

La proprietà è contenuta nel tipo del dato

Possibile preservare la proprietà con clone():
si effettua una copia del valore

Alcuni tipi copiano direttamente invece di trasferire (deep copy)
preservando la proprietà

Con i puntatori avviene un prestito - il riferimento NON ha proprietà, perciò quando viene chiamato drop ritorna al riferimento originale

Lifetime

Riferimenti sempre associati a memoria valida,
non liberata. No dangling pointers

