
Controllo dei tipi

Linguaggi e tipi

Il problema che ci poniamo ora è quello di comprendere come la struttura dei tipi del linguaggio possa influenzare il progetto e la struttura di implementazione dei linguaggi di programmazione.

L’importanza dei tipi anche nel Lambda Calcolo

Abbiamo visto che il lambda calcolo ha la potenza espressiva delle macchine di Turing
Possiamo codificare dati e strutture dati come espressioni del calcolo

$$True = \lambda t.\lambda f.t$$
$$False = \lambda t.\lambda f.f$$

$$0 = \lambda z.\lambda s.z$$
$$1 = \lambda z.\lambda s.s\ z$$

Dati e strutture dati possono essere codificati tramote opportune funzioni (i valori del calcolo)

L’importanza dei tipi per il lambda calcolo

Dato che nel lambda calcolo i programmi e i valori sono funzioni possiamo facilmente scrivere programmi che non sono corretti rispetto all’uso inteso dei valori

$$\boldsymbol{False\ 0 = (\lambda t.\lambda f.f)(\lambda z.\lambda s.z)}$$

Quale è l’errore?

L’importanza dei tipi per il lambda calcolo

Dato che nel lambda calcolo i programmi e i valori sono funzioni possiamo facilmente scrivre programmi che non corretti rispetto all’uso inteso dei valori

$$\boldsymbol{False\ 0 = (\lambda t.\lambda f.f)(\lambda z.\lambda s.z) \rightarrow \lambda f.f}$$

False si può effettivamente applicare 0 producendo un valore!!!!
(… non ha senso nella nostra interpretazione di False …)

I valori sono tutti rappresentati nello stesso modo (funzioni)
ma non si possono usare tutti nello stesso modo

… è solo un problema del lambda calcolo?

La stessa cosa accade nel linguaggio macchina

Le **istruzioni** sono parole della macchina: **sequenze di bit**

I **dati** sono codificati come parole macchina: **sequenze di bit**

Le **operazioni** prendono in ingresso **sequenze di bit** e restituiscono come risultato **sequenze di bit**

Cosa sono i tipi?

- Uno dei principi centrali dell'ingegneria del software è quello di realizzare opportuni **meccanismi** che permettano di ***rilevare precocemente*** **errori di programmazione** (es. False 0 è un errore di programmazione).
- I linguaggi di programmazione supportano questa idea con i sistema di tipo (e più in generale con gli strumenti di analisi statica).
- Il tipo è un **attributo** di un dato che descrive come il linguaggio di programmazione permetta di usare quel particolare dato.
  - I linguaggi di programmazione prevedono i tipi di dati di base come numeri interi (di varie dimensioni), numeri in virgola mobile (che approssimano i numeri reali), caratteri e booleani.
- **Un tipo limita i valori che un'espressione, come una variabile o una funzione, può assumere. Inoltre, definisce le operazioni che possono essere fatte sui dati e il modo in cui i valori di quel tipo possono essere memorizzati**.

Sistema dei tipi

- Un sistema dei tipi è un **metodo sintattico**, **effettivo** per **dimostrare l'assenza di comportamenti** anomali del programma **strutturando le operazioni del programma in base ai tipi di valori che calcolano**.

Sistema dei tipi

- Un sistema dei tipi è un **metodo sintattico**, **effettivo** per **dimostrare l'assenza di comportamenti** anomali del programma **strutturando le operazioni del programma in base ai tipi di valori che calcolano**.
- **Metodo sintattico:** la struttura sintattica guida il metodo di analisi del comportamento dei programmi

Sistema dei tipi

- Un sistema dei tipi è un **metodo sintattico**, **effettivo** per **dimostrare l'assenza di comportamenti** anomali del programma **strutturando le operazioni del programma in base ai tipi di valori che calcolano**.
- **Metodo sintattico:** la struttura sintattica guida il metodo di analisi del comportamento dei programmi
- **Effettivo**: si può definire **un algoritmo** che controlla i vincoli sui tipi e implementarlo in un compilatore o un interprete

Sistema dei tipi

- Un sistema dei tipi è un **metodo sintattico**, **effettivo** per **dimostrare l'assenza di comportamenti** anomali del programma **strutturando** le operazioni del programma in base ai tipi di valori che calcolano.
- **Metodo sintattico:** la struttura sintattica guida il metodo di analisi del comportamento dei programmi
- **Effettivo**: si può definire **un algoritmo** che controlla i vincoli sui tipi e implementarlo in un compilatore o un interprete
- **Strutturale**: i tipi assegnati alle componenti di un programma sono calcolati in modo ***composizionale:*** il tipo di un'espressione dipende solo dai tipi delle sue sottoespressioni

Sistema dei tipi

Un **sistema dei tipi** associa *tipi* ai valori calcolati

Esaminando il flusso dei valori calcolati, il sistema dei tipi tenta di dimostrare che non avvengano *errori di tipo*.

Il sistema stesso determina che cosa costituisce un errore di tipo, garantendo che le operazioni che si aspettano un certo tipo di valore non siano utilizzate con valori per i quali quell'operazione non ha senso.

Esempio

Supponiamo che l'espressione

```
if (condizione_complicata)
   { return "hello"/10; }
```

sia inserita all'interno di un programma di grosse dimensioni

Se i test del programma non forzano mai la valutazione <condizione_complicata>, l'errore nel ramo then (divisione di una stringa) non verrebbe mai individuato.
Tuttavia per un qualche input non considerato nella batteria dei test potrebbe accadere che venga eseguito il ramo then generando un errore di esecuzione.

Questo errore avrebbe potuto chiaramente essere evitato se il programmatore e/o l'implementazione del linguaggio avessero segnalato il valore "hello"/10 come qualcosa di inappropriato

Passiamo a JavaScript

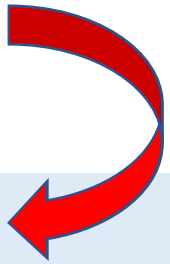
```
function addNumbers(x, y) {
   return x + y;}
// invocazione utente
console.log(addNumbers(3, "0"));
```

JavaScript Coercion:
l'operatore primitivo + può essere invocato con una coppia di valori (**number, string)**

Coercion: il valore numerico viene trasformato in un valore di tipo stringa.
viene stampato il valore "30" … una stringa.

Passiamo a TypeScript

```
function addNumbers(x: number, y: number) {
   return x + y;}
// invocazione utente
console.log(addNumbers(3, "0"));
```



TypeScript: segnala un errore di tipo rilevando questo bug

Type safety

La mancanza di **type safety** (correttezza dell'uso dei tipi) permette di scrivere programmi pieni di bug

Implicazione in cybersecurity:

- I bug sono vulnerabilità del software: un attaccante sfrutta i bug per alterare maliziosamente il comportamento del programma o per prendere il pieno controllo del flusso di esecuzione.

Controllo dei tipi

- **Statico** (in fase di compilazione) o **dinamico** (in fase di esecuzione),
  - Controllo statico trova gli errori prima di mandare il programma in esecuzione, non degrada le prestazioni
- Controllo di tipo (type checker) **verifica** che le intenzioni del programmatore (espresse dalle annotazioni di tipo) siano rispettate dal programma
- Un programma che supera il controllo dei tipi è garantito comportarsi bene in fase di esecuzione: non applica mai un'operazione ad un valore di tipo non corretto..
- **Controllo dei tipi: forma di correttezza parziale del programma (rispetto ai tipi, non rispetto ad altri tipi di errore)**

Domande significative

Come fa un progettista di linguaggio (o un programmatore) a sapere che programmi che superano il controllo dei tipi a run-time esibiscono le proprietà desiderate?

Per rispondere a questa domanda dobbiamo comprendere due aspetti essenziali
(1) come specificare i sistemi di tipi
(2) come dimostrare che un sistema di tipi è corretto.

Per dimostrare la correttezza è indispensabile comprendere nel dettaglio il significato dei programmi (cosa succede quando vengono eseguiti)
Comprendere i sistemi di tipo porterà ad una comprensione più profonda del significato dei programmi.

Un primo caso di studio
Sistema di tipo per espressioni

Espressioni e valori (sintassi)

Espressioni

E::=
- **true**
- **false**
- **NV**
- **if E then E else E**
- **succ E**
- **pred E**
- **isZero E**

Valori

V ::=
- **true**
- **false**
- **NV**

Valori numerici

NV ::=

0 | 1 | 2 | …

Regole di valutazione (interprete di espressioni)

$$if\ true\ then\ E1\ else\ E2 \rightarrow E1$$ **IF-TRUE**

$$if\ false\ then\ E1\ else\ E2 \rightarrow E2$$ **IF-FALSE**

$$\frac{E \rightarrow E'}{if\ E\ then\ E1\ else\ E2 \rightarrow if\ E'\ then\ E1\ else\ E2}$$ **IF-COND**

Regole di valutazione

$$\frac{E \to E'}{succ\ E \to succ\ E'} \qquad \frac{m = n + 1}{succ\ n \to m}$$

$$\frac{E \to E'}{pred\ E \to pred\ E'} \qquad \frac{n > 0 \qquad m = n - 1}{pred\ n \to m} \qquad pred\ 0 \to 0$$

$$\frac{E \to E'}{isZero\ E \to isZero\ E'} \qquad isZero\ 0 \to true \qquad \frac{n > 0}{isZero\ n \to false}$$

Tipi per espressoni

Il linguaggio delle espressioni prevede solo due **forme** sintattiche per i tipi: **Bool** o **Nat**

TIPI

T ::=
Bool
Nat

Controllo di tipo (type checker)

Il controllo di tipo definisce una relazione binaria (E, T) che associa il tipo T all'espressione E

Useremo la notazione E:T per indicare una coppia della relazione di assegnamento dei tipi.

Controllo dei tipi

Metodo sintattico

Regole definite per induzione strutturale sulla sintassi del programma

Regole di controllo dei tipi

$$true : Bool$$

$$false : Bool$$

$$n : Nat$$

$$\frac{E : Nat}{succ\ E : Nat}$$

$$\frac{E : Nat}{pred\ E : Nat}$$

$$\frac{E : Nat}{isZero\ E : Bool}$$

$$\frac{E : Bool, E1 : T, E2 : T}{if\ E\ then\ E1\ else\ E2 : T}$$

Controllo dei tipi: derivazione

$$\dfrac{\dfrac{}{???????}}{\mathbf{if\ isZero\ 0\ then\ 2\ else\ pred\ 1}:\mathbf{Nat}}$$

Ogni coppia (E, T) della relazione di tipo è caratterizzata da un albero di derivazione costruito da istanze delle regole di inferenza.

Controllo dei tipi: derivazione

$$\dfrac{\dfrac{\mathbf{0}:\boldsymbol{Nat}}{\boldsymbol{isZero}\ \mathbf{0}:\boldsymbol{Bool}} \quad \mathbf{2}:\boldsymbol{Nat} \quad \dfrac{\mathbf{1}:\boldsymbol{Nat}}{\boldsymbol{pred}\ \mathbf{1}:\boldsymbol{Nat}}}{\boldsymbol{if}\ \boldsymbol{isZero}\ \ \mathbf{0}\ \boldsymbol{then}\ \mathbf{2}\ \boldsymbol{else}\ \boldsymbol{pred}\ \ \mathbf{1}:\boldsymbol{Nat}}$$

Ogni coppia (E, T) della relazione di tipo è caratterizzata da un albero di derivazione costruito da istanze delle regole di inferenza.

Dalla teoria all'algoritmo di typechecking (pseudocodice)

```
typeof(E) = if (E = true) then Bool
            else if (E = false) then Bool
            else if (E = if E1 then E2 else E3) then {
                T1 = typeof(E1); T2 = typeof(E2); T3 = typeof(E3)
                if (T1 = Bool and T2=T3) then T2
                    else "type error" }
             else if (E = n) then Nat
             else if (E= succ E1) then {
                    T1 = typeof(E1)
                    if (T1 = Nat) then Nat else "type error" }
             else if (E = pred E1) then {
                    T1 = typeof(E1)
                    if (T1 = Nat) then Nat else "type error" }
            else if (E = iszero E1) then {
                    T1 = typeof(E1)
                    if T1 = Nat then Bool else "type error" }
```

Dalla teoria all'algoritmo di typechecking (pseudocodice)

```
typeof(E) = if (E = true) then Bool
            else if (E = false) then Bool
            else if (E = if E1 then E2 else E3) then {
                T1 = typeof(E1); T2 = typeof(E2); T3 = typeof(E3)
                if (T1 = Bool and T2=T3) then T2
                    else "type error" }
             else if (E = n) then Nat
             else if (E= succ E1) then {
                    T1 = typeof(E1)
                    if (T1 = Nat) then Nat else "type error" }
             else if (E = pred E1) then {
                    T1 = typeof(E1)
                    if (T1 = Nat) then Nat else "type error" }
            else if (E = iszero E1) then {
                    T1 = typeof(E1)
                    if T1 = Nat then Bool else "type error" }
```

$$true : Bool$$

$$false : Bool$$

$$\frac{E : Bool, E1 : T, E2 : T}{if\ E\ then\ E1\ else\ E2 : T}$$

$$n : Nat$$

$$\frac{E : Nat}{succ\ E : Nat}$$

$$\frac{E : Nat}{pred\ E : Nat}$$

$$\frac{E : Nat}{isZero\ E : Bool}$$

Precisione vs Approssimazione

I sistemi di tipo sono generalmente **imprecisi**: non definiscono esattamente quale tipo di valore sarà restituito da ogni programma, ma solo **un'approssimazione conservativa.**

Esempio

$$\frac{E: Bool, E1: T, E2: T}{if\ E\ then\ E1\ else\ E2: T}$$

Utilizzando questa regola non siamo in grado di associare un tipo all'espressione

$$if\ true\ then\ 0\ else\ false$$

Questa espressione sicuramente restituisce come risultato un valore numerico

Composizionalità

Osservazione:
La regola di controllo di tipo del condizionale richiede che le espressioni che costituiscono il "ramo then" e il "ramo else" abbiano lo stesso tipo, e che quel tipo sia il tipo del condizionale.

$$\frac{E : Bool, E1 : T, E2 : T}{if\ E\ then\ E1\ else\ E2 : T}$$

La regola garantisce la composizionalità.
Perché?

Composizionalità

Osservazione:
La regola di controllo di tipo del condizionale richiede che le espressioni che costituiscono il "ramo then" e il "ramo else" abbiano lo stesso tipo, e che quel tipo sia il tipo del condizionale.

$$\frac{E: Bool, E1: T, E2: T}{if\ E\ then\ E1\ else\ E2: T}$$

La regola garantisce la composizionalità.

$pred\ (if\ E\ then\ E1\ else\ E2)$

Un'espressione per essere usata in un certo contesto deve avere un tipo ben preciso

$if\ (if\ E\ then\ E1\ else\ E2)\ then\ E3\ else\ E4$

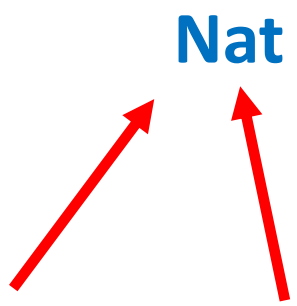
Composizionalità

Osservazione:
La regola di controllo di tipo del condizionale richiede che le espressioni che costituiscono il "ramo then" e il "ramo else" abbiano lo stesso tipo, e che quel tipo sia il tipo del condizionale.

$$\frac{E: Bool, E1: T, E2: T}{if\ E\ then\ E1\ else\ E2: T}$$

La regola garantisce la composizionalità.

Qualunque espressione usata come operando di pred deve avere tipo Nat



$pred\ (if\ E\ then\ E1\ else\ E2)$

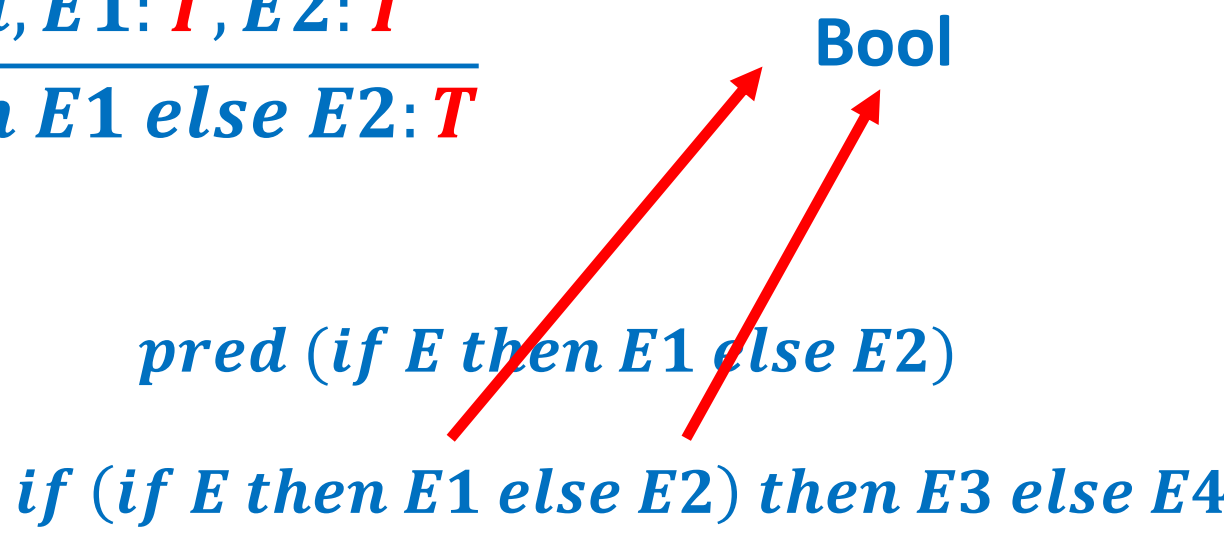
$if\ (if\ E\ then\ E1\ else\ E2)\ then\ E3\ else\ E4$

Composizionalità

Osservazione:
La regola di controllo di tipo del condizionale richiede che le espressioni che costituiscono il "ramo then" e il "ramo else" abbiano lo stesso tipo, e che quel tipo sia il tipo del condizionale.

$$\frac{E: Bool, E1: T, E2: T}{if\ E\ then\ E1\ else\ E2: T}$$

La regola garantisce la composizionalità.
Qualunque espressione usata come condizione di un if deve avere tipo Bool



Type Safety: Correttezza

La correttezza (type safety) del sistema di tipo
è espressa formalmente da queste due proprietà

Progresso: Se **E:T** allora **E** è un valore oppure $\boldsymbol{E \rightarrow E'}$
per una qualche espressione **E'**

Conservazione: Se **E:T** e $\boldsymbol{E \rightarrow E'}$ allora $\boldsymbol{E'}$**:T**

Type Safety: Correttezza

La correttezza (type safety) del sistema di tipo
è espressa formalmente da queste due proprietà

Progresso: Se **E:T** allora **E** è un valore oppure $E \rightarrow E'$
per una qualche espressione **E'**

Progresso: Una espressione ben tipata non si blocca a run-time (può fare un passo)

Conservazione: Se **E:T** e $E \rightarrow E'$ allora **E':T**

Conservazione: I tipi sono preservati dalle regole di esecuzione

Type Safety: Correttezza

La correttezza (type safety) del sist
è espressa formalmente da queste

Queste due proprietà, messe insieme, garantiscono che **l'intera esecuzione** dell'espressione non si blocchi a run-time

Progresso: Se **E:T** allora **E** è un valore oppure $E \rightarrow E'$ per una qualche espressione **E'**

Progresso: Una espressione ben tipata non si blocca a run-time (può fare un passo)

Conservazione: Se **E:T** e $E \rightarrow E'$ allora **E':T**

Conservazione: I tipi sono preservati dalle regole di esecuzione

Progresso

Progresso: Se **E:T** allora **E** è un valore oppure $E \rightarrow E'$
per una qualche espressione **E'**

Dimostrazione

Per induzione sulla struttura della derivazione di E:T

Casi base

$true : Bool$

$false : Bool$

$0 : Nat$

Immediato: true, false e 0 sono valori

Progresso

Progresso: Se **E:T** allora **E** è un valore oppure $E \rightarrow E'$
per una qualche espressione **E'**

Dimostrazione

Per induzione sulla struttura della derivazione di E:T

Casi Induttivi, vediamo solo (altri sono analoghi):

E = if E1 then E2 else E3:T

$$\frac{E1: Bool, E2: T, E3: T}{E = if\ E1\ then\ E2\ else\ E3: T}$$

Applicando al regola di tipo per il condizionale otteniamo

$$E1: Bool, E2: T, E3: T$$

Per ipotesi induttiva, E1 è un valore oppure esiste E4 tale che $E1 \rightarrow E4$

Nel caso sia un valore allora deve essere necessariamente true o false. In questo caso le regole della semantica di IF con condizioni true e false per fare il passo a E $\rightarrow E2$ oppure E $\rightarrow E3$.

Nel secondo caso ($E1 \rightarrow E4$) otteniamo che if E1 then E2 else E3 $\rightarrow$ if E4 then E2 else E3 applicando la terza regola della semantica dell'IF

Conservazione

Se **E:T** e $\boldsymbol{E \rightarrow E'}$ allora $\boldsymbol{E'}$**:T**

Dimostrazione. Solita induzione strutturale sulla derivazione di E:T.
Casi di base sono immediati (true, false e 0 sono valori, quindi non fanno nessun passo)

Conservazione

Se **E:T** e $E \to E'$ allora E'**:T**

Dimostrazione. Solita induzione strutturale sulla derivazione di E:T.
Casi induttivi
Supponiamo che l'ultima regola applicata sia la regola del condizionale.
E = if E1 then E2 else E3:T

$$\frac{E1: Bool, E2: T, E3: T}{if\ E1\ then\ E2\ else\ E3: T}$$

Pertanto abbiamo che

$$E1: Bool, E2: T, E3: T$$

E1 è un valore oppure, per ipotesi induttiva, esiste E4 tale che $E1 \to E4$
Nel caso sia un valore allora deve essere necessariamente true o false.
Nel caso sia true allora $E \to E2$ **e sappiamo già che E2:T.**
Il caso falso è simmetrico.
Nel secondo caso ($E1 \to E4$**) dobbiamo procedere come segue…**

Consideriamo il caso in cui $E1 \rightarrow E4$, quindi:

E= if E1 then E2 else E3 → *if E4 the E2 else E3*

Applicando l'ipotesi induttiva alla derivazione E1:Bool otteniamo che E4: Bool

Combinando questo risultato con le derivazioni di tipo (che valgono per ipotesi)

E2:T e E3:T

possiamo derivare applicando la regola IF

$$\frac{E4: Bool, E2: T, E3: T}{if\ E4\ then\ E2\ else\ E3: T}$$

Questo conclude il caso.

I casi delle altre regole di inferenza sono simili

Conclusione

- Abbiamo visto come strutturare una strategia di definizione e di verifica di un sistema di tipo in un caso semplice: il linguaggio delle espressioni.
- Passiamo ora ad un secondo caso di studio.