Tipi: mono, poly, classi

- Tipi
  - Monomorfi 'w' :: Char
    fact:: Int -> Int
  - Polimorfi flip:: (a -> b -> c) -> (b -> a -> c)
    lenght:: [a] -> Int
  - Constrained elem:: Eq a => a -> [a] -> Bool
    sort:: Ord a => [a] -> [a]
- Haskell:
  - Typed Language = alle variabili è associabile un TIPO (in un sistema di tipi non banale)
  - Explicitly Typed = i TIPI sono parte della sintassi (talvolta, parzialmente: non tutti esprimibili)
  - Statically checked = controllo dei tipi statico (simbolico: indipendente dai valori di input, una sola volta: per tutti i vari inputs)
  - Type soundness = tutti i programmi che superano il *check* non generano errori di tipo a run-time
- Vantaggi:

  Errori solo *trapped* (calcolo si blocca immediatamente - *untrapped* = errori che non si rilevano immediatamente:

  1) accedere indirizzi fuori dai bounds di un array, 2) trasferire controllo a una parola di memoria che non è una istruzione, 3) calcolare divisione per 0, 4) leggere un intero attendendo una stringa,

  sol. 1) untrapped, 2) untrapped, 3) trapped, 4) untrapped

  Documentazione: domini di valori coinvolti, relazioni tra domini e comportamenti generali attesi

  Locating: usare il tipo per localizzare una funzione tra quelle aventi un dato tipo {ad. Flip::(a->b->c)->b->a->c}

  Compilation: analisi del programma e generazione del codice semplificate

Sistema di tipi: Tipi, Regole, Derivazioni

- Tipi
  - **Sintassi**: Formalmente, introdotti attraverso un linguaggio context-free che ne fissa la sintassi
  - Esempio: T ::= k | T -> T
- Regole di tipizzazione:
  - **Sintassi**: $R \equiv (\Pi_1 \ldots \Pi_n) ==> \Pi$ **Grafica**: $\dfrac{\Pi_1 \ldots \Pi_n}{\Pi}$
    - Premises ==> Conclusion -- $R \equiv P(R) ==> C(R)$
    - (Judgment) $\Pi \equiv \Gamma \mid\!- \Im$
    - (ambiente di tipi) Γ: $\emptyset$, $x_1{:}T_1,\ldots,x_n{:}T_n$ *(xi variabili e Ti tipi)*
    - (formula) $\Im$: (*tipi* o) coppie *temine:tipo* che coinvolgono variabili nell'ambiente del jud.

Esempio: **(val +)**

$$\frac{\Gamma \mid\!- n{:}\ nat \qquad \Gamma \mid\!- m{:}\ nat}{\Gamma \mid\!- n+m{:}\ nat}$$

(val n) (n=0,..)

$$\frac{\Gamma \mid\!- \Diamond}{\Gamma \mid\!- n{:}\ nat}$$

- Derivazioni
  - **Sintassi:** Albero (rovesciato) di juds con archi orientati (Π_{up},Π_{down}) solo se $\exists$ R: $\Pi_{up} \in P(R)$ & $\Pi_{down} = C(R)$

$$\dfrac{\dfrac{\emptyset \mid\!- \Diamond}{\emptyset \mid\!- 5{:}\ nat}\,\textbf{Val 5} \qquad \dfrac{\emptyset \mid\!- \Diamond}{\emptyset \mid\!- 3{:}\ nat}\,\textbf{Val 3}}{\emptyset \mid\!- 5+3{:}\ nat}\,\textbf{Val +}$$

Monomorfo: F_1 esteso con prodotto

- **tipi:** A,B ::= k | A --> B | A + B | A x B | Unit *-- in Haskell Unit è ()*

  termini: M,N ::= x | \x:A->M | M N | (M,N) | first M | second M | unit
- **regole:**

(env ∅)
$$\frac{}{\emptyset \vdash \Diamond}$$

(env x)
$$\frac{\Gamma \vdash A \quad x \notin dom(\Gamma)}{\Gamma, x{:}A \vdash \Diamond}$$

(Tipo base)
$$\frac{\Gamma \vdash \Diamond \quad A \in k}{\Gamma \vdash A}$$

(val x)
$$\frac{\Gamma_1, x{:}A, \Gamma_2 \vdash \Diamond}{\Gamma_1, x{:}A, \Gamma_2 \vdash x{:}A}$$

(Tipo ->)
$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \dashrightarrow B}$$

(val fun)
$$\frac{\Gamma, x{:}A \vdash M{:}B}{\Gamma \vdash \backslash x{:}A\text{->}M{:}\ A \dashrightarrow B}$$

(val appl)
$$\frac{\Gamma \vdash M{:}\ A \dashrightarrow B \quad \Gamma \vdash N{:}\ A}{\Gamma \vdash M\ N{:}\ B}$$

(Tipo Unit)
$$\frac{\Gamma \vdash \Diamond}{\Gamma \vdash Unit}$$

(Val Unit)
$$\frac{\Gamma \vdash \Diamond}{\Gamma \vdash unit{:}\ Unit}$$

(Tipo prodotto)
$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \times B}$$

(Val prodotto)
$$\frac{\Gamma \vdash N{:}A \quad \Gamma \vdash M{:}B}{\Gamma \vdash (N,M){:}\ A \times B}$$

(val first)
$$\frac{\Gamma \vdash M{:}\ A \times B}{\Gamma \vdash first\ M : A}$$

(val second)
$$\frac{\Gamma \vdash M{:}\ A \times B}{\Gamma \vdash second\ M : B}$$

Checking Monomorfo: esempio

Provare che: ∅, y:U->U |- \z:U->y(z): U->U, per un tipo basico U∈k

(env ∅)

∅ |- ◊

(Tipo base)

Γ |- ◊   A∈k
Γ |- A

env ∅

∅|- ◊

U∈K

T.base

∅ |- U

env ∅

∅ |- ◊

T.base

∅ |- U

T.->

∅ |- U -> U

env x

∅, y: U -> U | - ◊

T.base

∅, y: U -> U | - U

U∈K

env x

∅, y: U -> U, z: U | - ◊

val x

∅, y: U -> U, z: U | - y: U->U

∅, y: U -> U, z: U | - ◊

val x

∅, y: U -> U, z: U | - z: U

val appl

∅, y: U -> U, z: U | - y(z): U

val fun

∅, y: U -> U | -\z:U -> y(z): U -> U

Type Checking: meccaniziamo la prova

- **Type checking:** processo di dimostrazione la cui conclusione (jud) asserisce quanto cercato

  ∅, y: U -> U | -\z:U -> y(z): U -> U

- **Type Checker:** meccanizzazione del processo di dimostrazione, procede anche attraverso semplificazioni (quali definizione di ambiente corretto)

```
________ env ∅          ________ env ∅
∅ |- ◊                  ∅ |- ◊
________ T.base         ________ T.base
∅ |- U                  ∅ |- U
________________________________ T.->
∅ |- U -> U
________________________________ env x
∅, y: U -> U | - ◊
________________________________ T.base
∅, y: U -> U | - U
________________________________ env x
∅, y: U -> U, z: U | - ◊                       ∅, y: U -> U, z: U | - ◊
________________________________ val x         ________________________________ val x
∅, y: U -> U, z: U | - y: U->U                 ∅, y: U -> U, z: U | - z: U
______________________________________________________________________________ val appl
∅, y: U -> U, z: U | - y(z): U
________________________________ val fun
∅, y: U -> U | -\z:U -> y(z): U -> U
```

Polimorfo: F_2 (tipi parametrici)

- **tipi:** A,B ::= X | A --> B | ∀X.A

  termini: M,N ::= x | \x:A->M | M N | ΔX.M | M A

  ΔT.\f: T-->T->\g: T-->T-> \x: T -> g(f g x)

regole:

(env ∅)
$$\frac{}{\varnothing \vdash \lozenge}$$

(env x)
$$\frac{\Gamma \vdash A \quad x \notin dom(\Gamma)}{\Gamma, x{:}A \vdash \lozenge}$$

(env X)
$$\frac{\Gamma \vdash \lozenge \quad X \notin dom(\Gamma)}{\Gamma, X \vdash \lozenge}$$

(Tipo X)
$$\frac{\Gamma_1, X, \Gamma_2 \vdash \lozenge}{\Gamma_1, X, \Gamma_2 \vdash X}$$

(val x)
$$\frac{\Gamma_1, x{:}A, \Gamma_2 \vdash \lozenge}{\Gamma_1, x{:}A, \Gamma_2 \vdash x{:}A}$$

(Tipo ->)
$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \to B}$$

(val fun)
$$\frac{\Gamma, x{:}A \vdash M{:}\ B}{\Gamma \vdash \backslash x{:}A{\to}M{:}\ A \to B}$$

(val appl)
$$\frac{\Gamma \vdash M{:}\ A \to B \quad \Gamma \vdash N{:}\ A}{\Gamma \vdash M\ N{:}\ B}$$

(Tipo ∀)
$$\frac{\Gamma, X \vdash A}{\Gamma \vdash \forall X.A}$$

(val fun2)
$$\frac{\Gamma, X \vdash M{:}\ A}{\Gamma \vdash \Delta X.M{:}\ \forall X.A}$$

(val appl2)
$$\frac{\Gamma \vdash M{:}\ \forall X.A \quad \Gamma \vdash B}{\Gamma \vdash M\ B{:}\ [B/X]A}$$

Checking Polimorfo: esempio

Provare che:

|- Δa. Δb. \y:a-->b -> \z: a -> y(z): ∀a. ∀b. (a-->b)-->a-->b

env ∅

∅ |- ◊   env X

∅, a |- ◊   env X

∅, a, b |- ◊   T. X   T. X

∅, a, b |- a   ∅, a, b |- b   T.->

∅, a, b |- a --> b   env x

∅, a, b, y: a --> b | - ◊   T. X

∅, a, b, y: a --> b | - a   env x

∅, a, b, y: a --> b, z:a | - ◊   val x   val x

∅, a, b, y: a --> b, z:a | - y: a --> b   ∅, a, b, y: a --> b, z:a | - z: a   val appl

∅, a, b, y: a --> b, z:a | - y(z): b   val fun

∅, a, b, y: a --> b | - \ z:a -> y(z): a --> b   val fun

∅, a, b| -\ y:a --> b -> \ z:a -> y(z): (a --> b) --> a --> b   val fun2

∅, a | - Δb. \ y:a --> b -> \ z:a -> y(z): ∀b. (a --> b) --> a --> b   val fun2

∅ | - Δa. Δb. \ y:a --> b -> \ z:a -> y(z): ∀a. ∀b. (a --> b) --> a --> b

Type Checking Polimorfo: unificazione

- **Problemi:**
  - unicità del tipo
  - sup (estremo superiore) se più di uno

- **Esempio**:

  h ≡ Δa. Δb. \g:(Int,[b]) --> Int -> \f: (a,Char) --> (a,[Char]) -> g . f

- **Le regole**:

  composizione e tipo lista e val lista
  quelle attese

- **Unificazione**:

  termini come tipi:
  t1 = (a,[Char]), t2 = (Int,[b])
  sostituzione come ambiente
  ρ = [Int/a, Char/b]
  most general instance come sup
  ρ(t1) = ρ(t2) = (Int,[Char])

Unificazione: algoritmo

- **Termini: u,v**
  - V-variabili: x,y
  - C-costruttori: c con arità ar(c)≥0
  - T-termni = V + {c(u1,…,un) | c∈C & ar(c)=n & u1,…,un ∈T}
- **Esempio**: (x,[Char]) con (_), [_], costruttori binari, Char costruttore 0-arity
- **Sostituzione**: ρ = <u1/x1,…,un/xn> oppure error con
  - dom(ρ)={x1,…,xn},
  - cod(ρ)=var(u1)+…+var(un) & var(x)={x}, var(c(u1,…,un))=var(u1)+…+var(un)
  - cod(ρ)∩dom(ρ)={} (idempotente)
- **istanza e composizione**: ρ = <u1/x1,…,un/xn> , σ = <v1/y1,…,vk/yk>
  - istanza: ρ(x)=u se u/x∈ρ   ρ(x)=x se x∉ dom(ρ)
    ρ(c(u1,…,un))=c(ρ(u1),…, ρ(un))
  - composizione: σ.ρ(u)= σ(ρ(u))
    σ.ρ = < σ(u1)/x1,…,σ(un)/xn, v1/y1,…,vk/yk> --- assunto: dom(ρ)∩dom(σ)={}
    σ.error = error.σ = error --- sempre (i.e. ∀ σ)
- **Algoritmo**:

```
unify t t' = []    se t = t'
unify x t = [t/x]    se x∉var(t)
unify x t = error   se x∈var(t)
unify t x = unify x t
unify c(u1,…,un) c(v1,…,vn) = unifyL <> [u1,…,un] [v1,…,vn]
unify _ _ = error
        where unifyL error _ _ = error
              unifyL ρ [] [] = ρ
              unifyL ρ (u:us) (v:vs) = unifyL σ σ(us) σ(vs)
                                       where σ = (unify ρ(u) ρ(v)) . ρ
```

Unificazione: applicazioni

- Unificare: Γ |- y:∀a.∀b. u1(a)-->u2(b) ed ancora, Γ |- z: ∀c.v1(c)

  Cosa possiamo concludere per A in --- Γ |- y(z) :A
  - Soluzione: A = ∀d.d
    - » mgu(u1(a)-->u2(b) , v1(c)-->d)=ρ
    - » mgi=ρ(u1(a)-->u2(b)), ρ(v1(c)), ρ(A)= ρ(d)
- Unificare. Nel modello sopra, si assuma y:∀b.(Int -> b) e y(z):A con A= Bool cosa dire del tipo di z?
- Unificare: le seguenti coppie di tipi mostrando sostituzione e mgi
  - (Int -> b), (a -> Bool)
  - (Int, a, a), (a, a, [Bool])
- Istanza: dimostrare che (Bool,[Bool]) è istanza di ciascuno dei seguenti tipi:
  - (a, [a]) -- mostrare la relativasostituzione
  - (b, c) -- mostrare la relativa sostituzione
- Verifica: se la funzione f::(a,[a]) -> b è applicabile ai seguenti termini:
  - (2,[3]) -- giustifica l'affermazione
  - (2,[])
  - (2,[True])

Type Checking con classi e overloading

- **tipi:** A,B ::= X | A-->B | ∀X.A | C => A --- estesi con C => A
- **regole:** generalizzate con constraints sulle variabili di tipo

(val fun)

$$\frac{\Gamma, x{:}C_A \Rightarrow A \mid\!- M{:}\ C_B \Rightarrow B}{\Gamma \mid\!- \backslash x\text{->}M{:}\ C_A \cup C_B \Rightarrow A\text{->}B}$$

- 1) **Unificazione**: C_A=>A con C_B=>B

  Esempio: Γ |- member: Eq a => [a] --> a --> Bool   Γ |- e: Ord b => [[b]]

  Quanto vale T in Γ |- member(e): T

mgu([a]-->a-->Bool, [[b]] --> d)

ρ =[[b]/a, [b]-->Bool/d]

- 2) **Controllo istanze di classe**: Eq a => [a]--> a --> Bool con Ord b => [[b]] --> d
  - Istanziamo d, uniamo i contsraints: ρ((Eq a, Ord b) => d) = (Eq [b], Ord b) => [b]-->Bool
  - Riduciamo e controlliamo:
    - Eq [b] non è un constraints (in Haskell)
    - rimpiazzabile con Eq b se
      - instance Eq b => Eq [b] where …. é presente nel programma

- 3) **Semplificazione**: (Eq b, Ord b) => [b]-->Bool
  - Riduciamo: Ord b => [b]-->Bool
    - se Class Eq a => Ord a where… è presente nel programma

Esercizi - 1:

- Definizione Normalizzata: di una definizione di funzione Haskell è l'equivalente definizione che rimpiazza equazioni con lambda-astratti.
  - a) si fornisca la normalizzata delle seguenti:
    - curry g x y = g (x , y)
    - atList 0 (x:xs) = x
      atList n (x:xs) = atList (n-1) xs
  - b) si calcoli il tipo di curry e quello atList.
- Tipi: Calcola il tipo dell'applicazione (f [] []), assunto f:: [a] -> [b] -> a -> b
- Tipi: Dato f con tipo come sopra, calcola il tipo della funzione h definita sotto:
  - h x = f x x
- Tipi: (craft2:13.8) Data la funzione curry sopra (ex. Def. Normalizzata) si calcoli il tipo per ciascuno dei termini sotto:
  - curry id - curry (curry id)
- Numerals di Church: si dia il tipo del numerale 1 e del numerale 2
- Numerals di Barendregt: si dia il tipo del numerale 2 e del numerale 3

Esercizi - 2:

- Numerals di Church: si dia il tipo delle funzioni succ e add sui numerali N_C
- Tipi: Utilizzando il sistema F1 si discuta il seguente asserto per tipi U,V∈k

  ∅ |- \f: U-->U ->\x:U -> f x x: V

- Tipi: Si discuta l'asserto seguente nel sistema F2

  ∅ |- \f:B-->C -> (\x:A -> f x x) (\x:A -> f x x)

- Haskell: Si consideri il seguente operatore di punto fisso:

  Y≡ \f -> \x -> (f x x) (f x x)

  - Si mostri che Y è tale che: per ogni F, Y F = F (Y F)
  - Si dica perché tale operatore non può essere espresso in Haskell
  - Si commenti la presenza della regola (val Y) nel sistema F1 con termini estesi come sotto

termini: M,N ::= x | \x:A->M | M N | Yx:A.M

(val Y)

$$\frac{\Gamma, x{:}A \vdash M{:}\ B}{\Gamma \vdash Yx{:}A.M{:}\ B}$$