

Izpeljava tipov

Kako programski jeziki uporabljajo tipe

Skoraj vsi programski jeziki imajo tipe, razlikujejo pa se po tem, kako se le-ti uporabljajo.

Kako striktni so tipi

Tipi so lahko bolj ali manj **striktni**. Če so popolnoma striktni, ima vsak izraz v veljavnem programu tip (OCaml, Standard ML, Haskell, Java, C++). Lahko se zgodi, da veljavni program nima tipa, ali vsaj ne takega, ki bi dobro opisal njegovo delovanje (Javascript, Python).

Primer: nabori v Pythonu imajo zelo ohlapen tip `tuple`, ki ne pove nič več kot to, da gre za urejeno večterico:

```
>>> type((1, 'foo', False))
<type 'tuple'>
```

V OCamlu so tipi striktni. Tip urejene trojice je bolj informativen:

```
# (1, "foo", false) ;;
- : int * string * bool = (1, "foo", false)
```

Dinamični in statični tipi

Poznamo delitev glede na *fazo*, v kateri se uporabijo tipi:

- Programski jezik ima **statične tipe**, če preveri ali izpelje tipe v *statični fazi*, se pravi ob prevajanju ali nalaganju koda, preden se koda požene. Primeri: C, C++, Java, C#, Standard ML, OCaml, Haskell, Swift, Scala.
- Programski jezik ima **dinamične tipe**, če preverja tipe v *dinamični fazi*, se pravi, ko se koda izvaja. Primeri: Scheme, Racket, Javascript, Python.

Preverjanje in izpeljevanje tipov

Programski jezik lahko tipe **preverja** ali **izpeljuje**:

- preverja** jih, če programer v večji meri zapiše tipe spremenljivk, funkcij in atributov, programski jezik pa preveri, da so pravino uporabljeni. Primeri: C, C++, Java, C#.
- izpeljuje** jih, če programerju ni treba podajati tipov spremenljivk, funkcij in atributov (lahko pa jih, če to želi), programski jezik pa sam ugotovi, kakšnega tipa so. Primeri: OCaml, Standard ML, Haskell.

Večina jezikov dopušča kombinacijo obeh tehnik. V OCamlu in Haskellu lahko predpišemo tip, čeprav bi ga programski jezik lahko izpeljal tudi sam. C++, Java in C# omogočajo izpeljavo tipov v omejenem obsegu, na primer pri tipih lokalnih spremenljivk.

Monomorfni in polimorfni tipi

Tipi so lahko:

- **monomorfni**, če ima vsak izraz največ en tip
- **polimorfni**, če ima lahko izraz hkrati več različnih tipov

Poznamo več vrst polimorfizma, danes bomo obravnavali **parametrični polimorfizem**.

Izpeljava tipov

Programski jeziki kot so OCaml, Standard ML in Haskell imajo polimorfne tipe, ki jih izpeljejo z algoritmom, ki sta ga razvila Hindley in Milner.

Kakšen tip ima funkcija $\lambda x. x$, oziroma v OCamlu `fun x -> x`? Možnih je veliko odgovorov:

- `int → int`
- `bool → bool`
- `int * int → int * int`
- $\alpha \text{ list} \rightarrow \alpha \text{ list}$ za poljuben α
- $\beta \rightarrow \beta$ za poljuben β .

Od vseh je zadnji najbolj splošen, ker lahko vse ostale dobimo tako, da **parameter** β zamenjamo s kakim drugim tipom. Pravimo, da je $\beta \rightarrow \beta$ *glavni* tip funkcije `fun x -> x`.

Definicija: Tip izraza je **glavni**, če lahko vse njegove tipe dobimo tako, da v glavnem tipu parametre zamenjamo s tipi (ki lahko vsebujejo nadaljne parametre).

OCaml je načrtovan tako, da ima vsak veljaven izraz glavni tip, ki ga OCaml izpelje sam. (Izjema so rekurzivne polimorfne funkcije, kjer mora programer sam opredeliti tip, saj algoritem za izračun glavnega tipa rekurzivne funkcije ne obstaja.)

Postopek izpeljave glavnega tipa

Glavni tip izraza e izpeljemo v dveh fazah:

1. Izračunamo kandidata za tip e , ki vsebuje neznanke, in enačbe, ki jim morajo neznanke zadoščati.
2. Rešimo enačbe s postopkom *združevanja*.

Druga faza se lahko zalomi, če se izkaže, da enačbe nimajo rešitve.

Prva faza

V prvi fazi izračunamo kandidata za tip in nabiramo enačbe, ki morajo veljati:

- `true` ima tip `bool`, brez enačb
- `false` ima tip `bool`, brez enačb
- celoštevilska konstanta `0, 1, 2, ...` ima tip `int`, brez enačb
- spremenljivka ima svoj dani tip (tipe spremenljivk sproti beležimo v *kontekstu*)
- aritmetični izraz $e_1 + e_2$:
 - izračunamo tip τ_1 izraza e_1 in dobimo še enačbe E_1

- izračunamo tip τ_2 izraza e_2 in dobimo še enačbe E_2

Tip izraza $e_1 + e_2$ je int , z enačbami E_1, E_2 in $\tau_1 = \text{int}, \tau_2 = \text{int}$. Podobno obravnavamo ostale aritmetične izraze $e_1 * e_2, e_1 - e_2, \dots$

- boolov izraz $e_1 \ \&\& \ e_2$: obravnavamo podobno kot aritmetični izraz, le da uporabimo pričakovani bool namesto int .
- primerjava celih števil $e_1 < e_2$:
 - izračunamo tip τ_1 izraza e_1 in dobimo še enačbe E_1
 - izračunamo tip τ_2 izraza e_2 in dobimo še enačbe E_2

Tip izraza $e_1 < e_2$ je bool , z enačbami E_1, E_2 in $\tau_1 = \text{int}, \tau_2 = \text{int}$

- pogojni stavek $\text{if } e_1 \text{ then } e_2 \text{ else } e_3$:
 - izračunamo tip τ_1 izraza e_1 in dobimo še enačbe E_1
 - izračunamo tip τ_2 izraza e_2 in dobimo še enačbe E_2
 - izračunamo tip τ_3 izraza e_3 in dobimo še enačbe E_3

Tip izraza $\text{if } e_1 \text{ then } e_2 \text{ else } e_3$ je τ_2 , z enačbami $E_1, E_2, E_3, \tau_1 = \text{bool}, \tau_2 = \tau_3$

- urejeni par (e_1, e_2) :
 - izračunamo tip τ_1 izraza e_1 in dobimo še enačbe E_1
 - izračunamo tip τ_2 izraza e_2 in dobimo še enačbe E_2

Tipi izraza (e_1, e_2) je $\tau_1 \times \tau_2$, z enačbami E_1, E_2 .

- prva projekcija $\text{fst } e$:
 - izračunamo tip τ izraza e in dobimo še enačbe E

Uvedemo nova parametra α in β (se ne pojavljata v E). Tip izraza $\text{fst } e$ je α , z enačbami $E, \tau = \alpha \times \beta$.

- druga projekcija $\text{snd } e$:
 - izračunamo tip τ izraza e in dobimo še enačbe E

Uvedemo nova parametra α in β . Tip izraza $\text{snd } e$ je β , z enačbami $E, \tau = \alpha \times \beta$.

- funkcija $\text{fun } x \rightarrow e$: uvedemo nov parameter α in zabeležimo, da ima x tip α , ter
 - izračunamo tip τ izraza e (pri predpostavki, da ima x tip α) in dobimo še enačbe E

Tip funkcije $\text{fun } x \rightarrow e$ je $\alpha \rightarrow \tau$ z enačbami E

- aplikacija $e_1 \ e_2$:
 - izračunamo tip τ_1 izraza e_1 in dobimo še enačbe E_1
 - izračunamo tip τ_2 izraza e_2 in dobimo še enačbe E_2

Uvedemo nov parameter α . Tip izraza $e_1 \ e_2$ je α , z enačbami $E_1, E_2, \tau_1 = \tau_2 \rightarrow \alpha$

- rekurzivna definicija $x = e$ (kjer se x pojavi v e): uvedemo nov parameter α , zabeležimo, da ima x tip α , ter

- izračunamo tip τ izraza e (pri predpostavki, da ima x tip α) in dobimo še enačbe E

Tip izraza x je τ , z enačbami E , $\alpha = \tau$. Opomba: običajno na ta način definiramo rekurzivne funkcije, torej bo x v resnici funkcija.

Druga faza: združevanje

Imamo množico enačb E

$l_1 = d_1$
 $l_2 = d_2$
 $l_3 = d_3$
 $\dots$
 $l_i = d_i$

v neznankah $\alpha, \beta, \gamma, \delta, \dots$ Rešujemo z naslednjim postopkom:

1. Imamo seznam rešitev r , ki je na začetku prazen.
2. Če je E prazna množica, vrnemo rešitev r .
3. Sicer iz E odstranimo katerokoli enačbo $l = d$ in jo obravnavamo:
 - če sta leva in desna stran povsem enaki, enačbo zavržemo ter gremo na korak 2
 - če je enačba oblike $\alpha = d$, kjer je α neznanka:
 - če se α pojavi v d , postopek prekinemo, ker *ni rešitve*
 - sicer smo našli rešitev za α , namreč $\alpha \mapsto d$. Povsod v r in E zamenjamo α z d in v r dodamo rešitev $\alpha \mapsto d$
 - če je enačba oblike $l = \alpha$, kjer je α neznanka, imamo primer, ki je simetričen prejšnjemu
 - če je enačba oblike $(l_1 \rightarrow l_2) = (d_1 \rightarrow d_2)$, v E dodamo enačbi $l_1 = d_1$ in $l_2 = d_2$ in gremo na korak 2
 - če je enačba oblike $(l_1 \times l_2) = (d_1 \times d_2)$, v E dodamo enačbi $l_1 = d_1$ in $l_2 = d_2$ in gremo na korak 2
 - če je enačba katerekoli druge oblike, na primer $(l_1 \rightarrow l_2) = (d_1 \times d_2)$, postopek prekinemo, ker *ni rešitve*.

Kako to deluje, si pogledjmo na primerih.

Primer 1

Izpelji glavni tip funkcije

```
fun x -> x + 3
```

Odgovor: `int -> int`

Primer 3

Izpelji glavni tip izraza

```
if 3 < 5 then (fun x -> x) else (fun y -> y + 3)
```

Odgovor: `int -> int`

Churchovi numerali

Kakšen je tip števila 3?

```
0 = (λ f x . x)
1 = (λ f x . f x)
2 = (λ f x . f (f x))
3 = (λ f x . f (f (f x)))
```

To naj izračuna OCaml:

```
let zero  = (fun f x -> x) ;;
let one   = (fun f x -> f x) ;;
let two   = (fun f x -> f (f x)) ;;
let three = (fun f x -> f (f (f x))) ;;
```

Churchovi-Scottovi numerali

Kakšen je tip števila 3?

```
0 = (λ f x . x)
1 = (λ f x . f 0 x)
2 = (λ f x . f 1 (f 0 x))
3 = (λ f x . f 2 (f 1 (f 0 x)))
```

To naj izračuna OCaml: