

# Type Inference

## Típuskövetkeztetés

Bagossy Attila

Deklaratív programozás  
2019. május 06.





# Bevezetés

Típus

Erős és gyenge típusosság

Statikus és dinamikus típusosság



# Miért kellene típusok?

---

- A hardver csupán biteket ismer.
- Egy bitsorozat jelenthet például
  - egész számot,
  - lebegőpontos számot,
  - memóriacímet,
  - végrehajtható utasítást.
- Alacsony szinten a programozó dolga az ezek közti különbségtétel.



# Miért kellene típusok? (2)

---

## IMPLICIT KONTEXTUS

A típusok implicit kontextust rendelnek az egyes műveletekhez, így azt a programozónak nem kell explicit megjelölnie.





# Miért kellene típusok? (2)

---

## IMPLICIT KONTEXTUS

A típusok implicit kontextust rendelnek az egyes műveletekhez, így azt a programozónak nem kell explicit megjelölnie.

## SZABÁLYOS MŰVELETEK KIJELÖLÉSE

A típusok pontosan kijelölik, hogy melyek azok a műveletek, amelyek egy szemantikailag helyes programban megjelenhetnek.



# Típus

---

## DENOTÁCIÓS JELENTÉS

A típus értékek halmaza.



# Típus

---

## DENOTÁCIÓS JELENTÉS

A típus értékek halmaza.

## KONSTRUKTÍV JELENTÉS

A típus valamilyen primitív (beépített) típus, avagy egy konstruktorral képzett összetett (kompozit) típus.



# Típus

---

## DENOTÁCIÓS JELENTÉS

A típus értékek halmaza.

## KONSTRUKTÍV JELENTÉS

A típus valamilyen primitív (beépített) típus, avagy egy konstruktorral képzett összetett (kompozit) típus.

## ABSZTRAKCIÓS JELENTÉS

A típus egy interfész, melyet a támogatott műveletek alkotnak.





# Típusellenőrzés

---

- Angolul *type checking*.
- Annak ellenőrzése, hogy egy program megfelel az adott nyelv típusrendszerének.
- Ha a program egy része nem megfelelően típusozott, akkor típusütközés vagy angolul *type clash* keletkezik.



# Típusosság

---

## MIKOR ELLENŐRIZZÜK A TÍPUSOKAT?

Statikus és dinamikus típusosság.

Static és dynamic typing.

## MENNYIRE SZIGORÚ A TÍPUSRENDSZER?

Gyenge és erős típusosság.

Weak és strong typing.



# Gyenge típusosság – C

---

To this day, many C programmers believe that strong typing just means pounding extra hard on the keyboard.

Peter van der Linden





# Típuskövetkeztetés

Miért jó a típuskövetkeztetés?  
Hindley-Milner típusrendszer

# Típuskövetkeztetés és -ellenőrzés

---

## TÍPUSKÖVETKEZTETÉS

Adott term esetén meghatározza, hogy a term típusozható-e, és amennyiben igen, akkor automatikusan egy típust rendel hozzá.

## TÍPUSELLENŐRZÉS

Típusannotációkkal ellátott termről eldönti, hogy megfelelően típusozott vagy sem.





# Példa – Összeadás

---

```
function addTwo(x: int): int
{
    return 2 + x;
}
```



# Példa – Összeadás

---

```
function addTwo(x: int): int
{
    return 2 + x;
}
```



# Példa – Összeadás

---

```
function addTwo(x: int): int
{
    return 2 + x;
}
```

Tegyük fel, hogy ismert a + operátor szignatúrája:

```
function operator+(a: int, y: int): int
```



# Példa – Összeadás

---

```
function addTwo(x)
{
    return 2 + x;
}
```

Tegyük fel, hogy ismert a + operátor szignatúrája:

```
function operator+(a: int, y: int): int
```

Ekkor az addTwo függvény szignatúrája automatikusan meghatározható.



# Példa – Összeadás

---

```
function addTwo(x)
{
    return 2 + x;
}
```

A típuskövetkeztetés nem jelent dinamikus típusosságot.

Bár a típusannotációk hiányoznak, azonban minden típus fordítási időben ismert.





# Miért jó a típuskövetkeztetés?

---

- Javítja a forráskód olvashatóságát.
  - Csökken a szintaktikai zaj, amelyet a rengeteg típusannotáció okoz.
- Növeli a produktivitást.
  - A programozónak nem kell a típusannotációk elhelyezésével foglalkoznia.
- Segíti a fejlesztést.
  - Az automatikusan kikövetkeztetett típus hibákat jelezhet előre.



# Hindley-Milner következtetés

- Alfred Tarski már az 1920-as években alkalmazott valamiféle illesztést.
- Carew Meredith már az 1950-es években használt egy hasonló algoritmust.
- Típuskövetkeztetést Haskell Curry is leírt 1958-ban az egyszerű típusos  $\lambda$ -kalkulusra.
- Roger Hindley 1967-ben írta le Curry algoritmusának kibővített változatát.
- Robin Milner 1978-ban fedezte fel/találta fel újra.
- Luis Damas 1982-ben bizonyította az algoritmus teljességét.



# Hindley-Milner következtetés

---

- Mindig a legáltalánosabb típust határozza meg.
  - *Principal vagy most general type.*
- Nem szükséges típusannotációk hozzáadása, azokat az algoritmus automatikus meghatározza.
- Futási ideje a legtöbb, valós programra lineáris a program méretével.



# Algoritmus

---

1. Szintaktikai elemzéssel építsünk fel egy absztrakt szintaxisfát (AST).
2. A fa leveleihez rendeljünk típusváltozókat.
3. Készítsünk egy típus-egyenletrendszert:
  - A literálok, beépített operátorok, beépített függvények, már típusozott termék típusai ismertként fognak megjelenni.
  - A fa felépítésének megfelelően adunk hozzá további egyenleteket.
4. Illesztéssel oldjuk meg az egyenletrendszert.
5. Határozzuk meg a legfelső szintű típust.



# Példa – Összeadás

---

```
function addTwo(x)
{
    return 2 + x;
}
```

$\text{addTwo} = x \Rightarrow 2 + x$





# Legáltalánosabb típus

---

Ugyanazon term többféleképpen is típusozható.

`length [] = 0`

`length xs:x = 1 + length xs`



# Legáltalánosabb típus

---

Ugyanazon term többféleképpen is típusozható.

```
length [] = 0
```

```
length xs:x = 1 + length xs
```

```
[int] -> int
```

```
[String] -> int
```



# Legáltalánosabb típus

---

Ugyanazon term többféleképpen is típusozható.

```
length [] = 0
```

```
length xs:x = 1 + length xs
```

```
[int] -> int
```

```
[String] -> int
```

```
[a] -> int
```



# Típusellenőrzés – Formálisan

$\overline{e \vdash x : A}$  if  $e$  contains  $x : A$

$$\frac{e \vdash u : A \quad e \vdash t : A \rightarrow B}{e \vdash t u : B}$$

$$\frac{(e, x : A) \vdash t : B}{e \vdash \text{fun } x \rightarrow t : A \rightarrow B}$$

$$\overline{e \vdash n : \text{nat}}$$

$$\frac{e \vdash u : \text{nat} \quad e \vdash t : \text{nat}}{e \vdash t \otimes u : \text{nat}}$$

$$\frac{e \vdash t : \text{nat} \quad e \vdash u : A \quad e \vdash v : A}{e \vdash \text{ifz } t \text{ then } u \text{ else } v : A}$$

$$\frac{(e, x : A) \vdash t : A}{e \vdash \text{fix } x \text{ } t : A}$$

$$\frac{e \vdash t : A \quad (e, x : A) \vdash u : B}{e \vdash \text{let } x = t \text{ in } u : B}$$



# Típuskövetkeztetés – Formálisan

$$\frac{}{e \vdash x \rightsquigarrow A, \emptyset} \text{ if } e \text{ contains } x : A$$

$$\frac{e \vdash u \rightsquigarrow A, E \quad e \vdash t \rightsquigarrow B, F}{e \vdash t \ u \rightsquigarrow X, E \cup F \cup \{B = A \rightarrow X\}}$$

$$\frac{(e, x : X) \vdash t \rightsquigarrow A, E}{e \vdash \text{fun } x \rightarrow t \rightsquigarrow (X \rightarrow A), E}$$

$$\frac{}{e \vdash n \rightsquigarrow \text{nat}, \emptyset}$$

$$\frac{e \vdash u \rightsquigarrow A, E \quad e \vdash t \rightsquigarrow B, F}{e \vdash t \otimes u \rightsquigarrow \text{nat}, E \cup F \cup \{A = \text{nat}, B = \text{nat}\}}$$

$$\frac{e \vdash t \rightsquigarrow A, E \quad e \vdash u \rightsquigarrow B, F \quad e \vdash v \rightsquigarrow C, G}{e \vdash \text{ifz } t \text{ then } u \text{ else } v \rightsquigarrow B, E \cup F \cup G \cup \{A = \text{nat}, B = C\}}$$

$$\frac{(e, x : X) \vdash t \rightsquigarrow A, E}{e \vdash \text{fix } x \ t \rightsquigarrow A, E \cup \{X = A\}}$$

$$\frac{e \vdash t \rightsquigarrow A, E \quad (e, x : A) \vdash u \rightsquigarrow B, F}{e \vdash \text{let } x = t \text{ in } u \rightsquigarrow B, E \cup F}$$





# Felhasznált irodalom

---

- Gilles Dowek and Jean-Jacques Lvy. 2010. *Introduction to the Theory of Programming Languages* (1st ed.). Springer Publishing Company, Incorporated.
- Peter Selinger. 2008. *Lecture Notes on the Lambda Calculus*. Dalhousie University, Halifax, Canada.
- Michael L. Scott. 2009. *Programming Language Pragmatics, Third Edition* (3rd ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.



# Felhasznált irodalom (2)

---

- Daniel P. Friedman and Mitchell Wand. 2008. *Essentials of Programming Languages, 3rd Edition* (3 ed.). The MIT Press.
- Mooly Sagiv. *Types and Type Inference*.  
<http://www.cs.tau.ac.il/~msagiv/courses/apl12/types.pdf>



# Köszönöm a figyelmet!

Az előadás elérhető a következő címen:

[git.io/fjn58](https://git.io/fjn58)

