Type Inference
Típuskövetkeztetés

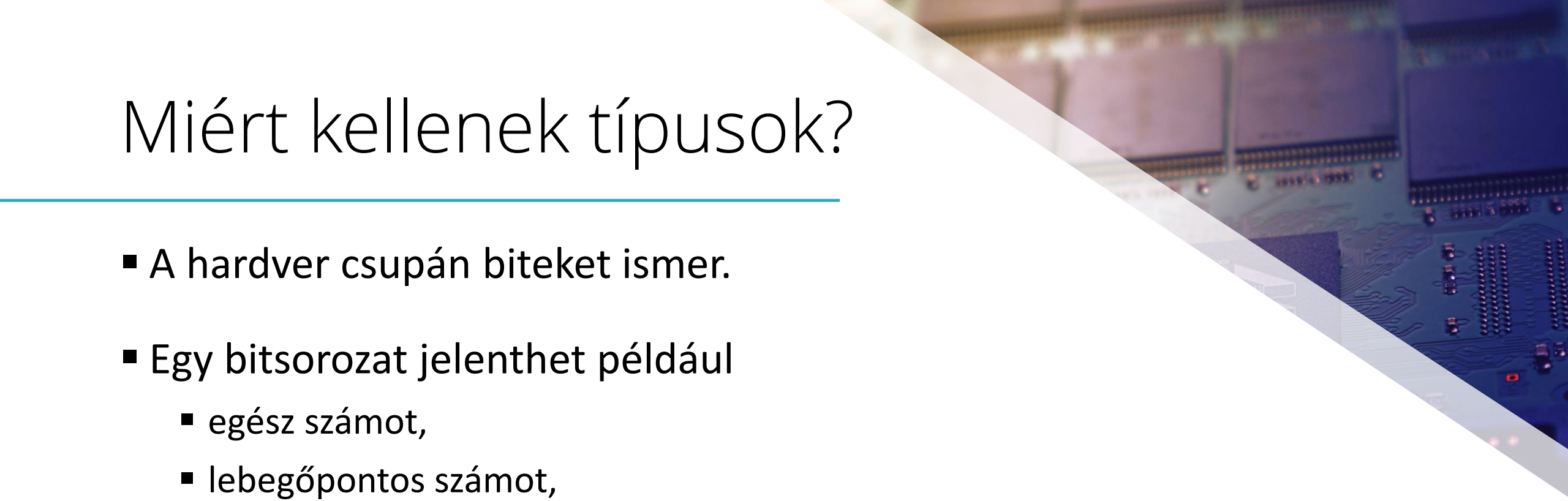
Bagossy Attila

Deklaratív programozás
2019. május 06.

Bevezetés

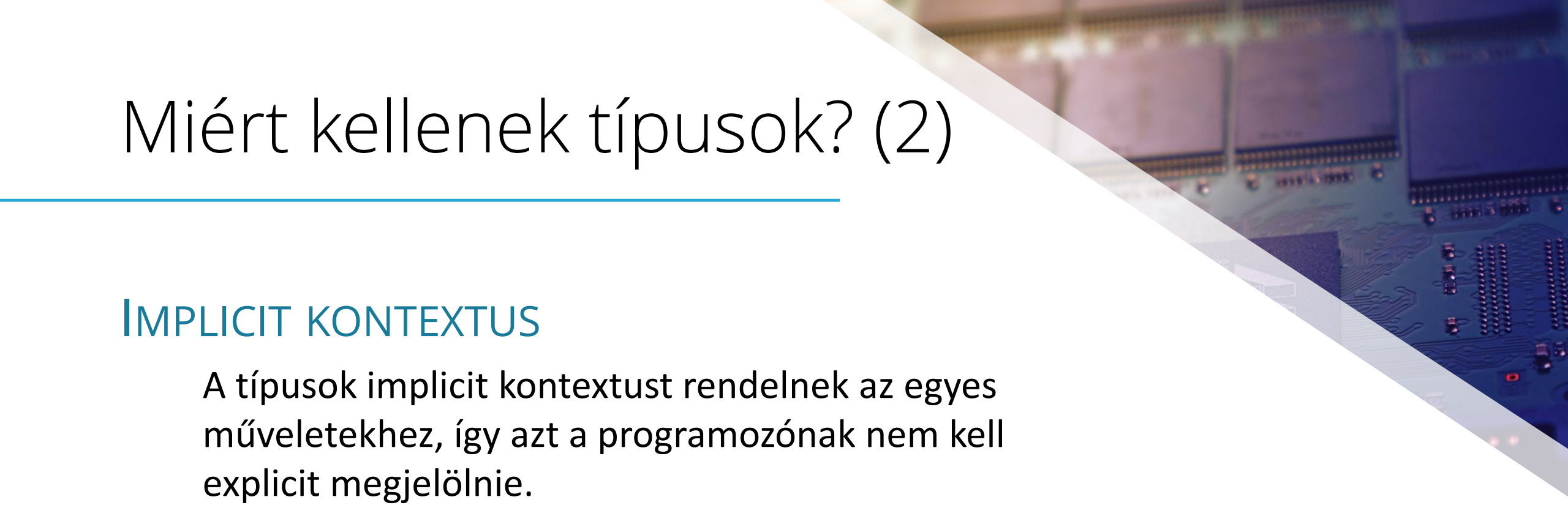
Típus
Erős és gyenge típusosság
Statikus és dinamikus típusosság

Miért kellenek 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.

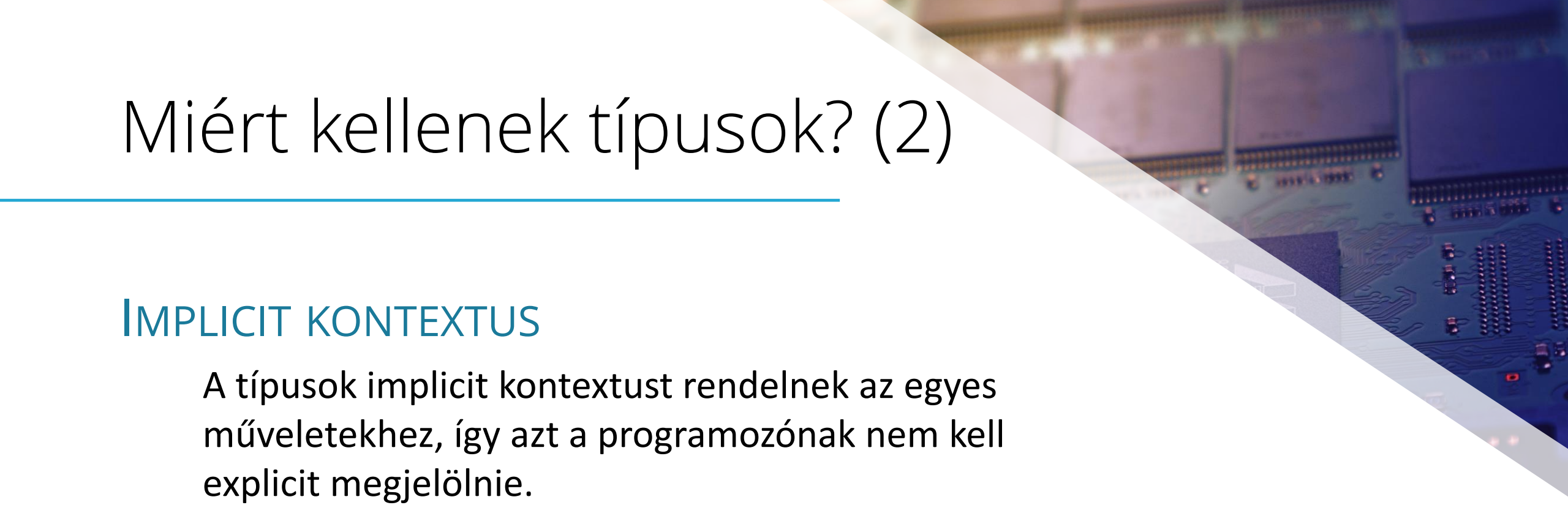
Miért kellenek típusok? (2)



Implicit kontextus

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

Miért kellenek 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

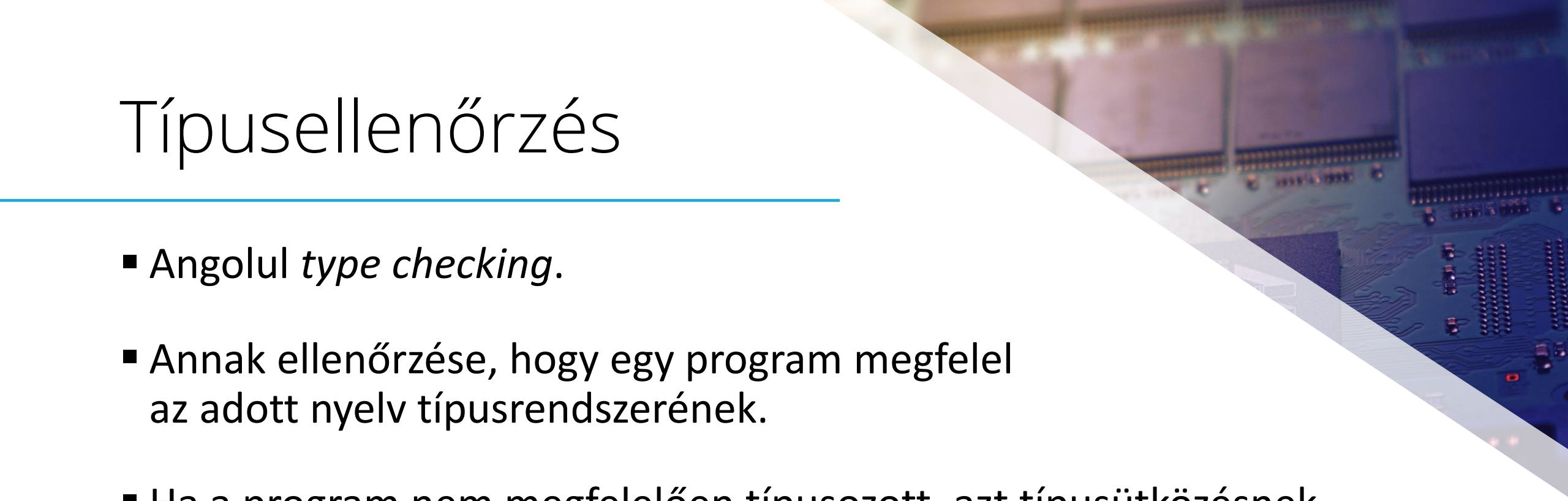
Denotációs jelentés

A típus értékek halmaza.

Konstruktív jelentés

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

Típusellenőrzés



- Angolul *type checking*.
- Annak ellenőrzése, hogy egy program megfelel az adott nyelv típusrendszerének.
- Ha a program nem megfelelően típusozott, azt típusütközésnek vagy angolul *type clash*nek nevezzük.

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.

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;
}
```

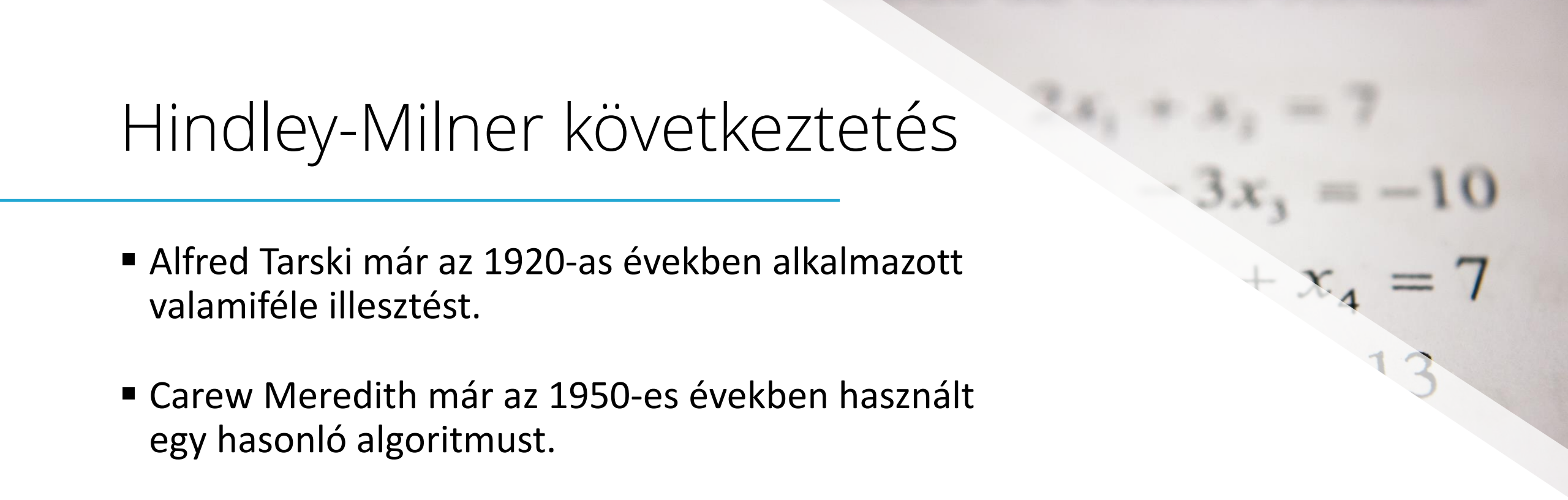
A típuskövetkeztetés nem jelent dinamikus típusosságot.

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 λ-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 termek 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;
}
```

```
addTwo = x => 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

$$\frac{}{\texttt{e} \vdash \texttt{x} : \texttt{A}} \text{ if e contains } \texttt{x} : \texttt{A}$$

$$\frac{\texttt{e} \vdash \texttt{u} : \texttt{A} \quad \texttt{e} \vdash \texttt{t} : \texttt{A} \texttt{ -> } \texttt{B}}{\texttt{e} \vdash \texttt{t u} : \texttt{B}}$$

$$\frac{\texttt{(e, x : A)} \vdash \texttt{t} : \texttt{B}}{\texttt{e} \vdash \texttt{fun x -> t} : \texttt{A -> B}}$$

$$\frac{}{\texttt{e} \vdash \texttt{n} : \texttt{nat}}$$

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

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

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

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

Típuskövetkeztetés – Formálisan

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

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

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

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

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

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

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

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

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

Felhasznált irodalom (2)

- *git.io/fjn58*