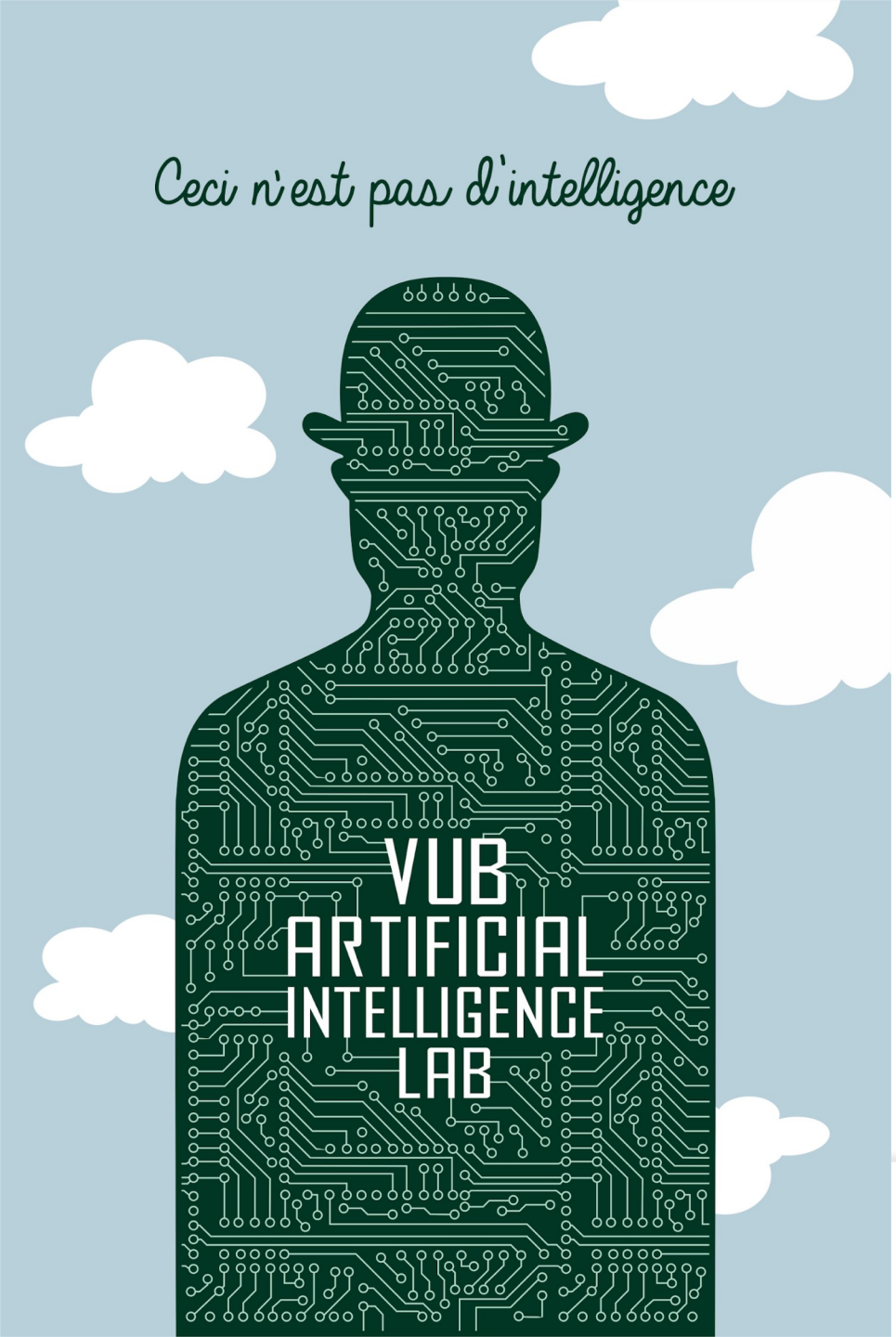

Logica en formele systemen

Lambda Calculus

Inleiding en basisbegrippen

Prof. dr. Marjon Blondeel

Academiejaar 2024-2025

Inhoud lambda calculus

- Inleiding
- Basisbegrippen
- Rekenen met lambda expressies
- Fixpunten en recursie

Inleiding

Lambda calculus (λ-calculus) geïntroduceerd in de jaren 1930 als een system om het begrip functie abstract the bestuderen.

Ook om begrip berekenbaarheid te formaliseren: welke functies zijn al dan niet berekenbaar waarbij een functie berekenbaar is desda er een programma P bestaat dat de functie berekent: Als P stopt voor een input a dan is dat de correcte output en anders stopt P nooit.

Inleiding

Historisch gezien is het beslissen of twee λ-expressies equivalent zijn het eerste probleem waarvan werd aangetoond dat het onbeslisbaar is.

Het is onmogelijk om een programma te schrijven dat, gegeven twee willekeurige λ-expressies bepaalt of ze al dan niet equivalent zijn.

λ-calculus en programmeertalen

Formalisme werd ontwikkeld voor het bestaan van concrete programmeertalen, maar vormt de basis van functionele programmeertalen zoals Lisp en Scheme.

Voorbeelden van invloed:

- call by name: parameters worden pas geëvalueerd op moment dat ze gebruikt worden
- hogere orde functies: output functie mag functie zijn, parameter mag functie zijn

Functies bekeken op een andere manier

functie	1 argument
$f(x) = 2 + x$	$+_2(x) = x + 2$
$f(x, y) = x + y$	$+_y(x) = x + y$

$$+_z\left(+_y(x)\right) = (x + y) + z$$

Functies bekeken op een andere manier

$+_2(x) = x + 2$	$+(2) = +_2$	$+(2)(x) = +_2(x) = x + 2$
$+_y(x) = x + y$	$+(y) = +_y$	$+(y)(x) = +_y(x) = x + y$

Merk op: output van $+(y)$ is een functie

Functies: λ-notatie

Voorbeeld: $f(x) = 2 + x = +(2)(x)$

$$\lambda x.\big((+)2\big)x$$

Vorm: λ <parameter>.<voorschrift>

Toepassing functie: (functie)argument

Functies: λ-notatie

Voorbeeld: $f(x) = 2 + x$

$$\lambda x.\big((+)2\big)x$$

- we hoeven geen naam te geven aan een functie
- functies hoeven maar 1 argument te hebben
  - meerdere parameters kunnen we simuleren (output mag terug een functie zijn)

Voorbeelden Scheme

```
(define (add-three number) (+ number 3))

(define (add-three-to-each lst) (map add-three lst))

(define (add-three-to-each lst)
        (map (lambda (number) (+ number 3)) lst))
```

Inhoud lambda calculus

- Inleiding
- Basisbegrippen
- Rekenen met lambda expressies
- Fixpunten en recursie

Abstractie en applicatie

2 fundamentele bewerkingen in λ-calculus

- abstractie: aanmaken van een functievoorschrift
- applicatie: aanroepen van een functievoorschrift

Lambda expressies (syntax)

Zij V een verzameling variabelen. De verzameling van λ-expressies Λ wordt alsvolgt gedefinieerd:

1. $V \subseteq \Lambda$
2. Als $M \in \Lambda$ en $\mathrm{N} \in \Lambda$ dan is $(M)N \in \Lambda$. (applicatie)
3. Als $x \in V$ en $M \in \Lambda$ dan is $\lambda x.M \in \Lambda$. (abstractie)
4. Niets anders zit in Λ.

Definitie

Lambda expressies: intuïtie

1. $V \subseteq \Lambda$: elke variabele is een λ-expressie
2. $(M)N \in \Lambda$ correspondeert met een functieaanroep: *(functie)parameter*
3. $\lambda x.M \in \Lambda$ correspondeert met een functievoorschrift: *λformeleparameter.functievoorschrift*

$(\lambda x.x)y$: pas functie $(\lambda x.x)$ toe op parameter y

$(\lambda x.x)$: eerste $\mathbf{x}$ is formele parameter, tweede is voorschrift

Lambda expressies: opmerkingen

- geen constanten in de definitie
  - later tonen we dat we bv gehele getallen dmv λ-expressies kunnen voorstellen
- We spreken van “functies” maar er bestaan geen functies in λ-calculus, enkel goed gevormde expressies waar wij een betekenis aan geven.

Lambda expressies: voorbeeld

Stel we hebben variabelen x, y, u, f. Dan zijn volgende expressies λ-expressies:

x

$\lambda x.\, x$

$\lambda x.\, \lambda y.\, (y)x$

$(\lambda y.\, (x)y)\lambda x.\, (u)x$

$\lambda x.\, \lambda y.\, x$

$\lambda f.\, \lambda x.\, (f)x$

$\lambda f.\, \lambda x.\, (f)(f)x$

Volgorde van functieapplicatie

Functieapplicatie gebeurt van rechts naar links

$(P)(Q)x$: eerst Q toepassen op x en dan P toepassen op het resultaat, in “gewone” notatie: $P(Q(x))$

$((P)Q)x$: eerst P toepassen op Q en dan de resulterende λ-expressie toepassen op x

Merk op: $((P)(Q))x$ is geen geldige λ-expressie

Currying: inleiding

Uit de definitie volgt dat een “functie” maar 1 parameter kan hebben. Hoe kunnen we functies met meerder variabelen dan voorstellen?

idee: $f: A \times B \rightarrow C$ vervangen door $g: A \rightarrow (B \rightarrow C)$ waarbij $g(a) = f_a$ en $f_a(b) = f(a, b)$

voorbeeld: de plus functie definiëren we als $+(a) = +_a$ en $+_a(b) = a + b$

Currying: mechanisme

Wat met n parameters? Techniek “currying” vernoemd naar Haskell B. Curry.

$A_1 \times \cdots \times A_n \to B$ kunnen we herleiden naar $A_1 \to A_2 \times \cdots \times A_n \to B$

indien $n > 2$:

$A_2 \times \cdots \times A_n \to B$ kunnen we herleiden naar $A_2 \to A_3 \times \cdots \times A_n \to B$

enz, uiteindelijk krijgen we

$$A_1 \to (A_2 \to \cdots (A_n \to B))$$

Currying: voorbeeld

Vier simpele functies met elk 1 argument

- $+(y) = +_y$
- $+_y(x) = x + y$
- $\times(y) = \times_y$
- $\times_y(x) = x \cdot y$

Combineren tot complexere functies

$$\times(3)(+(2)x) = \times_3(+(2)x) = \times_3(+_2(x)) = \times_3(x+2)$$
$$= (x+2) \cdot 3$$

Currying in Scheme

```
(lambda (x y) (+ x y))
currying:
(lambda (x)
        (lambda (y)
                (+ x y)))
```

```
((lambda (x)
        (lambda (y)
                (+ x y))) 3)
->
(lambda (y)
                (+ 3 y)))
```

Currying in Scheme

```
(lambda (x y) (+ x y))
currying:
(lambda (x)
        (lambda (y)
                (+ x y)))
```

```
(((lambda (x)
        (lambda (y)
                (+ x y))) 3)4)
->
((lambda (y)
                (+ 3 y)))4)
```

Currying in React

```jsx
import React, { useState } from 'react';

function MyComponent() {
  const [state, setState] = useState({ name: '', email: '', notes: '' });

  const handleChange = (fieldName) => (event) => {
    const { value } = event.target;
    setState((prevState) => ({
      ...prevState,
      [fieldName]: value,
    }));
  };

  return (
    <div>
      <input
        type="text"
        placeholder="Enter your name"
        value={state.name}
        onChange={handleChange('name')}
      />
      <input
        type="text"
        placeholder="Enter your email"
        value={state.email}
        onChange={handleChange('email')}
      />
      <textarea
        placeholder="Enter your notes"
        value={state.notes}
        onChange={handleChange('notes')}
      />
      <button>Submit</button>
    </div>
  );
}

export default MyComponent;
```

Currying voor λ-expressies

Beschouw de volgende λ-expressie G

$$\lambda x.\lambda f.(f)x$$

We kunnen dit zien als een functie G van 2 variabelen (in "gewone" notatie):

$$G(x, f) = f(x)$$

nadeel: leesbaarheid

Substitutie: waarom bestuderen?

Herschrijven van λ-expressies (toepassen van voorschrift op actuele parameter)

- $(\lambda x.\, x)y$ kunnen we herschrijven als y
- $(\lambda x.\, (x)x)y$ kunnen we herschrijven als $(y)y$
- $(\lambda x.\, \lambda y.\, x)y$?

vrije variabele die gebonden wordt!

Binding en bereik

Voor een λ-expressie $\lambda x.M$ zegt men dat

- λx een binding is van x in M
- het bereik van de binding M is: alle (nog niet gebonden) voorkomens van x in $\lambda x.M$ zijn gebonden

In een λ-expressie heten alle voorkomens van variabelen die niet gebonden zijn vrij

Vrije variabelen

De vrije variabelen van een λ-expressie zijn alsvolgt gedefinieerd:

- $\forall x \in V: VV(x) = \{x\}$
- $\forall M, N \in \Lambda: VV\big(M(N)\big) = VV(M) \cup VV(N)$
- $\forall x \in V, \forall M \in \Lambda: VV(\lambda x.M) = VV(M) \setminus \{x\}$

Notatie: $VV(M)$ van vrije variabelen van M.

Definitie

Gesloten λ-expressie

Een λ-expressie zonder vrije variabelen heet gesloten of een combinator. De verzameling van combinatoren wordt genoteerd als Λ_0.

Definitie

Vrije/gebonden variabelen: voorbeelden

- $VV(\lambda w.v) = \{v\}$
- $VV(\lambda v.v) = \emptyset$
- $VV(\lambda v.w) = \{w\}$
- $VV((\lambda v.v)v) = \{v\}$

Vrije/gebonden variabelen: voorbeelden

$\lambda x.\,(x)y$

- vrij: y
- gebonden: x

$(\lambda x.\,x)\lambda y.\,(y)x$

- vrij: laatste x
- gebonden: eerste x en y

Merk op: een variabele kan vrij en gebonden voorkomen in een λ-expressie

Substitutie: intuïtief

Intuitief willen we een functieaanroep uitwerken. Bijvoorbeeld in

$$(\lambda x.M)P$$

willen we elke formele parameter x die hoort bij de binding λx vervangen door de actuele parameter P.

Notatie: $[P/x]M$ alle voorkomens van x in M vervangen door P

Let op! We willen niet dat vrije variabelen in P gebonden worden.

Substitutie: definitie

Beschouw λ-expressies P en M en $x \in V$. De substitutie $[P/x]M$ van P voor x in M wordt alsvolgt gedefinieerd:

(S1) $[P/x]x = \mathrm{P}$

(S2) $[P/x]y = y$ als $y \in V \setminus \{x\}$

(S3) $[P/x](F)Q = ([P/x]F)[P/x]Q$

(S4) $[P/x]\lambda x.M = \lambda x.M$

(S5) $[P/x]\lambda y.M = \lambda y.[P/x]M$ als $y \neq x$ en $y \notin VV(P)$

(S6) $[P/x]\lambda y.M = \lambda z.[P/x][z/y]M$ als $y \neq x$ en $\mathrm{z} \notin VV(P)$ en $y \in VV(P)$

Definitie

Substitutie: definitie intuïtief (1/4)

(S1) $[P/x]x = \mathrm{P}$

triviaal: in x moeten we x vervangen door P

(S2) $[P/x]y = y$ als $y \in V \setminus \{x\}$

triviaal: in y moeten we x vervangen door P, maar er is helemaal geen x $(y \neq x)$

Substitutie: definitie intuïtief (2/4)

(S3) $[P/x](F)Q = ([P/x]F)[P/x]Q$

applicatie: we voeren de substitutie door op beide delen

(S4) $[P/x]\lambda x.M = \lambda x.M$

we moeten alle x vervangen door P, maar alle x zijn gebonden door de λx, we doen dus niets

Substitutie: definitie intuïtief (3/4)

(S5) $[P/x]\lambda y.M = \lambda y.[P/x]M$ als $y \neq x$ en $y \notin VV(P)$

indien y niet vrij voorkomt in P, laten we de λy staan voeren we de substitutie door in M

wat als y wel vrij voorkomt in P? dan zou het kunnen dat een oorspronkelijke vrije y opeens gebonden zou worden door de λy die vooraan komt, bv

$$[\lambda u.(y)u/x]\lambda y.(x)y$$

Substitutie: definitie intuïtief (4/4)

(S6) $[P/x]\lambda y.M = \lambda z.[P/x][z/y]M$ als $y \neq x$ en $\mathbf{z} \notin VV(P)$ en $y \in VV(P)$

we herschrijven $\lambda y.M$ naar $\lambda z.M$ waarbij we alle y in M vervangen door z, d.i. $[z/y]M$, uiteraard kiezen we $\mathbf{z} \notin VV(P)$, vervolgens kunnen we (S5) toepassen

Substitutie: voorbeelden (1/2)

- $[u/x]\lambda u.\, x =_{(s6)} \lambda z.\, [u/x][z/u]x =_{(s2)} \lambda z.\, [u/x]x =_{(s1)} \lambda z.\, u$
- $[u/x]\lambda u.\, u =_{(s6)} \lambda z.\, [u/x][z/u]u =_{(s1)} \lambda z.\, [u/x]z =_{(s2)} \lambda z.\, z$
- $[u/x]\lambda y.\, x =_{(s5)} \lambda y.\, [u/x]x =_{(s1)} \lambda y.\, u$
- $[u/x]\lambda y.\, u =_{(s5)} \lambda y.\, [u/x]u =_{(s2)} \lambda y.\, u$

Substitutie: voorbeelden (2/2)

$$
\begin{aligned}
&[\lambda u.\,(y)u/x]\lambda y.\,(x)y \\
=_{(s6)}\ &\lambda z.\,[\lambda u.\,(y)u/x][z/y](x)y \\
=_{(s3)}\ &\lambda z.\,[\lambda u.\,(y)u/x]([z/y]x)[z/y]y \\
=_{(s2)}\ &\lambda z.\,[\lambda u.\,(y)u/x](x)[z/y]y \\
=_{(s1)}\ &\lambda z.\,[\lambda u.\,(y)u/x](x)z \\
=_{(s3)}\ &\lambda z.\,([\lambda u.\,(y)u/x]x)[\lambda u.\,(y)u/x]z \\
=_{(s1)}\ &\lambda z.\,(\lambda u.\,(y)u)[\lambda u.\,(y)u/x]z \\
=_{(s2)}\ &\lambda z.\,(\lambda u.\,(y)u)z
\end{aligned}
$$