Синтаксис на безтиповото λ-смятане

Трифон Трифонов

λ-смятане и теория на доказателствата, 2018/19 г.

5 март 2019 г.

История на λ-смятането

(1928) Alonzo Church създава λ-смятането като алтернатива на теориите на типовете (Russel) и на множествата (Zermelo)

История на λ-смятането

(1928) Alonzo Church създава λ-смятането като алтернатива на теориите на типовете (Russel) и на множествата (Zermelo)

(1932) първата статия за λ-смятането като вид безтипова логика

История на λ-смятането

(1928) Alonzo Church създава λ-смятането като алтернатива на теориите на типовете (Russel) и на множествата (Zermelo)

(1932) първата статия за λ-смятането като вид безтипова логика

(1932) Kurt Gödel публикува теоремите си за непълнота

История на λ-смятането

(1928) Alonzo Church създава λ-смятането като алтернатива на теориите на типовете (Russel) и на множествата (Zermelo)

(1932) първата статия за λ-смятането като вид безтипова логика

(1932) Kurt Gödel публикува теоремите си за непълнота

(1933) кодиране на естествени числа в λ-смятането, подобрена версия

История на λ-смятането

(1928) Alonzo Church създава λ-смятането като алтернатива на теориите на типовете (Russel) и на множествата (Zermelo)

(1932) първата статия за λ-смятането като вид безтипова логика

(1932) Kurt Gödel публикува теоремите си за непълнота

(1933) кодиране на естествени числа в λ-смятането, подобрена версия

(1935) Kleene и Rosser откриват парадокс в λ-смятането като логическа система

История на λ-смятането

(1928) Alonzo Church създава λ-смятането като алтернатива на теориите на типовете (Russel) и на множествата (Zermelo)

(1932) първата статия за λ-смятането като вид безтипова логика

(1932) Kurt Gödel публикува теоремите си за непълнота

(1933) кодиране на естествени числа в λ-смятането, подобрена версия

(1935) Kleene и Rosser откриват парадокс в λ-смятането като логическа система

(1936) теорема на Church-Rosser за конфлуентност

История на λ-смятането

(1928) Alonzo Church създава λ-смятането като алтернатива на теориите на типовете (Russel) и на множествата (Zermelo)

(1932) първата статия за λ-смятането като вид безтипова логика

(1932) Kurt Gödel публикува теоремите си за непълнота

(1933) кодиране на естествени числа в λ-смятането, подобрена версия

(1935) Kleene и Rosser откриват парадокс в λ-смятането като логическа система

(1936) теорема на Church-Rosser за конфлуентност

(1936) Church доказва неразрешимост на предикатното смятане

История на λ-смятането

(1928) Alonzo Church създава λ-смятането като алтернатива на теориите на типовете (Russel) и на множествата (Zermelo)

(1932) първата статия за λ-смятането като вид безтипова логика

(1932) Kurt Gödel публикува теоремите си за непълнота

(1933) кодиране на естествени числа в λ-смятането, подобрена версия

(1935) Kleene и Rosser откриват парадокс в λ-смятането като логическа система

(1936) теорема на Church-Rosser за конфлуентност

(1936) Church доказва неразрешимост на предикатното смятане

(1937) Alan Turing дефинира първия комбинатор за намиране на неподвижна точка

История на λ-смятането

(1928) Alonzo Church създава λ-смятането като алтернатива на теориите на типовете (Russel) и на множествата (Zermelo)

(1932) първата статия за λ-смятането като вид безтипова логика

(1932) Kurt Gödel публикува теоремите си за непълнота

(1933) кодиране на естествени числа в λ-смятането, подобрена версия

(1935) Kleene и Rosser откриват парадокс в λ-смятането като логическа система

(1936) теорема на Church-Rosser за конфлуентност

(1936) Church доказва неразрешимост на предикатното смятане

(1937) Alan Turing дефинира първия комбинатор за намиране на неподвижна точка

(1940) Church дефинира типовото λ-смятане

Развитие на λ-смятането

(1960) John McCarthy създава първия функционален език LISP

Развитие на λ-смятането

(1960) John McCarthy създава първия функционален език LISP

(1958) Kurt Gödel дефинира системата T — разширение на типовото λ-смятане

Развитие на λ-смятането

(1960) John McCarthy създава първия функционален език LISP

(1958) Kurt Gödel дефинира системата T — разширение на типовото λ-смятане

(1966) Corrado Böhm и Wolf Gross създават CUCH, първият програмен език директно базиран на λ-смятането

Развитие на λ-смятането

(1960) John McCarthy създава първия функционален език LISP

(1958) Kurt Gödel дефинира системата T — разширение на типовото λ-смятане

(1966) Corrado Böhm и Wolf Gross създават CUCH, първият програмен език директно базиран на λ-смятането

(1969) Dana Scott дава семантичен модел на λ-смятането

Развитие на λ-смятането

(1960) John McCarthy създава първия функционален език LISP

(1958) Kurt Gödel дефинира системата T — разширение на типовото λ-смятане

(1966) Corrado Böhm и Wolf Gross създават CUCH, първият програмен език директно базиран на λ-смятането

(1969) Dana Scott дава семантичен модел на λ-смятането

(1969) William Howard развива идеите на Haskell Curry за съответствие между доказателства и типизирани λ-термове

Развитие на λ-смятането

(1960) John McCarthy създава първия функционален език LISP

(1958) Kurt Gödel дефинира системата T — разширение на типовото λ-смятане

(1966) Corrado Böhm и Wolf Gross създават CUCH, първият програмен език директно базиран на λ-смятането

(1969) Dana Scott дава семантичен модел на λ-смятането

(1969) William Howard развива идеите на Haskell Curry за съответствие между доказателства и типизирани λ-термове

(1971) Henk Barendregt дефинира понятието нерешими λ-термове

Развитие на λ-смятането

(1960) John McCarthy създава първия функционален език LISP

(1958) Kurt Gödel дефинира системата T — разширение на типовото λ-смятане

(1966) Corrado Böhm и Wolf Gross създават CUCH, първият програмен език директно базиран на λ-смятането

(1969) Dana Scott дава семантичен модел на λ-смятането

(1969) William Howard развива идеите на Haskell Curry за съответствие между доказателства и типизирани λ-термове

(1971) Henk Barendregt дефинира понятието нерешими λ-термове

(1972) Nicolaas Govert de Bruijn създава техника за избягване на проблеми при субституция

Развитие на λ-смятането

(1960) John McCarthy създава първия функционален език LISP

(1958) Kurt Gödel дефинира системата T — разширение на типовото λ-смятане

(1966) Corrado Böhm и Wolf Gross създават CUCH, първият програмен език директно базиран на λ-смятането

(1969) Dana Scott дава семантичен модел на λ-смятането

(1969) William Howard развива идеите на Haskell Curry за съотвествие между доказателства и типизирани λ-термове

(1971) Henk Barendregt дефинира понятието нерешими λ-термове

(1972) Nicolaas Govert de Bruijn създава техника за избягване на проблеми при субституция

(1972) Jean-Yves Girard дефинира системата F, полиморфно λ-смятане

λ-смятането в информатиката

(1975) Guy Steele и Gerald Sussman създават Scheme, диалект на LISP

λ-смятането в информатиката

(1975) Guy Steele и Gerald Sussman създават Scheme, диалект на LISP

(1977) John Backus (авторът на FORTRAN) популяризира функционалния стил

λ-смятането в информатиката

(1975) Guy Steele и Gerald Sussman създават Scheme, диалект на LISP

(1977) John Backus (авторът на FORTRAN) популяризира функционалния стил

(1980) Robin Milner създава функционалния език ML с полиморфни типове

λ-смятането в информатиката

(1975) Guy Steele и Gerald Sussman създават Scheme, диалект на LISP

(1977) John Backus (авторът на FORTRAN) популяризира функционалния стил

(1980) Robin Milner създава функционалния език ML с полиморфни типове

(1985) David Turner създава Miranda, първият комерсиален чист функционален език

λ-смятането в информатиката

(1975) Guy Steele и Gerald Sussman създават Scheme, диалект на LISP

(1977) John Backus (авторът на FORTRAN) популяризира функционалния стил

(1980) Robin Milner създава функционалния език ML с полиморфни типове

(1985) David Turner създава Miranda, първият комерсиален чист функционален език

(1990) Публикувана е първата версия на Haskell

λ-смятането в информатиката

(1975) Guy Steele и Gerald Sussman създават Scheme, диалект на LISP

(1977) John Backus (авторът на FORTRAN) популяризира функционалния стил

(1980) Robin Milner създава функционалния език ML с полиморфни типове

(1985) David Turner създава Miranda, първият комерсиален чист функционален език

(1990) Публикувана е първата версия на Haskell

(1998) Отваряне на кода на реализацията на Erlang

λ-смятането в информатиката

(1975) Guy Steele и Gerald Sussman създават Scheme, диалект на LISP

(1977) John Backus (авторът на FORTRAN) популяризира функционалния стил

(1980) Robin Milner създава функционалния език ML с полиморфни типове

(1985) David Turner създава Miranda, първият комерсиален чист функционален език

(1990) Публикувана е първата версия на Haskell

(1998) Отваряне на кода на реализацията на Erlang

(1990–2000) Функционални елементи в императивни езици: Python (1991), JavaScript (1995), Ruby (1995), ActionScript (1998)

λ-смятането в информатиката

(1975) Guy Steele и Gerald Sussman създават Scheme, диалект на LISP

(1977) John Backus (авторът на FORTRAN) популяризира функционалния стил

(1980) Robin Milner създава функционалния език ML с полиморфни типове

(1985) David Turner създава Miranda, първият комерсиален чист функционален език

(1990) Публикувана е първата версия на Haskell

(1998) Отваряне на кода на реализацията на Erlang

(1990–2000) Функционални елементи в императивни езици: Python (1991), JavaScript (1995), Ruby (1995), ActionScript (1998)

(2000–) Функционалният стил на програмиране превзема света: Scala (2003), F# (2005), C# (2007), Clojure (2007), C++11 (2011), Elixir (2011), Java 8 (2014)

Синтаксис на λ-смятането

Нека считаме, че разполагаме с изброим набор от синтактични променливи V, които означаваме с малки латински букви $x, y, z, \ldots$.

Синтаксис на λ-смятането

Нека считаме, че разполагаме с изброим набор от синтактични променливи V, които означаваме с малки латински букви $x, y, z, \ldots$. Азбуката на λ-смятането се състои от всички променливи, символа λ и скоби.

Синтаксис на λ-смятането

Нека считаме, че разполагаме с изброим набор от синтактични променливи V, които означаваме с малки латински букви $x, y, z, \ldots$. Азбуката на λ-смятането се състои от всички променливи, символа λ и скоби.

Дефиниция (λ-термове)

Дефинираме множеството от λ-термове Λ:

1. Ако $x \in V$, то $x \in \Lambda$ (*променлива*).
2. Ако $M, N \in \Lambda$, то $(MN) \in \Lambda$ (*апликация* или *прилагане* на терма M към терма N).
3. Ако $x \in V$ и $M \in \Lambda$, то $(\lambda_x M) \in \Lambda$ (*абстракция* или *построяване* на функция с аргумент x и тяло M).

Улесняващи нотации

- $A := B$ — синтактичният обект A се дефинира чрез израза B

Улесняващи нотации

- $A := B$ — синтактичният обект A се дефинира чрез израза B
- $A \equiv B$ — A и B съвпадат като синтактични обекти

Улесняващи нотации

- $A := B$ — синтактичният обект A се дефинира чрез израза B
- $A \equiv B$ — A и B съвпадат като синтактични обекти
- $A = B$ — A и B са равни в дадена логическа система

Улесняващи нотации

- $A := B$ — синтактичният обект A се дефинира чрез израза B
- $A \equiv B$ — A и B съвпадат като синтактични обекти
- $A = B$ — A и B са равни в дадена логическа система
- ще пропускаме най-външните скоби

Улесняващи нотации

- $A := B$ — синтактичният обект A се дефинира чрез израза B
- $A \equiv B$ — A и B съвпадат като синтактични обекти
- $A = B$ — A и B са равни в дадена логическа система
- ще пропускаме най-външните скоби
- ще записваме $(\ldots((M_1 M_2) M_3) \ldots M_k)$ съкратено като $M_1 \ldots M_k$ или като $\vec{M}$

Улесняващи нотации

- $A := B$ — синтактичният обект A се дефинира чрез израза B
- $A \equiv B$ — A и B съвпадат като синтактични обекти
- $A = B$ — A и B са равни в дадена логическа система
- ще пропускаме най-външните скоби
- ще записваме $(\ldots((M_1 M_2)M_3)\ldots M_k)$ съкратено като $M_1 \ldots M_k$ или като $\vec{M}$
- ще записваме $(\ldots((MN_1)N_2)\ldots N_k)$ съкратено като $MN_1 \ldots N_k$ или като $M\vec{N}$

Улесняващи нотации

- $A := B$ — синтактичният обект A се дефинира чрез израза B
- $A \equiv B$ — A и B съвпадат като синтактични обекти
- $A = B$ — A и B са равни в дадена логическа система
- ще пропускаме най-външните скоби
- ще записваме $(\ldots((M_1 M_2)M_3)\ldots M_k)$ съкратено като $M_1 \ldots M_k$ или като $\vec{M}$
- ще записваме $(\ldots((MN_1)N_2)\ldots N_k)$ съкратено като $MN_1 \ldots N_k$ или като $M\vec{N}$
- ще записваме $(\lambda_{x_1}(\lambda_{x_2}(\lambda_{x_3}\ldots(\lambda_{x_n}M)\ldots)))$ съкратено като $\lambda_{x_1,x_2,\ldots,x_n}M$ или просто $\lambda_{\vec{x}}M$

Улесняващи нотации

- $A := B$ — синтактичният обект A се дефинира чрез израза B
- $A \equiv B$ — A и B съвпадат като синтактични обекти
- $A = B$ — A и B са равни в дадена логическа система
- ще пропускаме най-външните скоби
- ще записваме $(\ldots((M_1 M_2)M_3)\ldots M_k)$ съкратено като $M_1 \ldots M_k$ или като $\vec{M}$
- ще записваме $(\ldots((MN_1)N_2)\ldots N_k)$ съкратено като $MN_1 \ldots N_k$ или като $M\vec{N}$
- ще записваме $(\lambda_{x_1}(\lambda_{x_2}(\lambda_{x_3}\ldots(\lambda_{x_n}M)\ldots)))$ съкратено като $\lambda_{x_1,x_2,\ldots,x_n}M$ или просто $\lambda_{\vec{x}}M$
- ще считаме, че апликацията е със по-висок приоритет от абстракцията, т.е. $\lambda_x(MN) \equiv \lambda_x MN \not\equiv (\lambda_x M)N$

Свободни и свързани променливи

Дефиниция (Свободни променливи)

Дефинираме функцията $\mathrm{FV} : \Lambda \to \mathcal{P}(V)$:

1. $\mathrm{FV}(x) := \{x\}$
2. $\mathrm{FV}(M_1 M_2) := \mathrm{FV}(M_1) \cup \mathrm{FV}(M_2)$
3. $\mathrm{FV}(\lambda_x N) := \mathrm{FV}(N) \setminus \{x\}$

Свободни и свързани променливи

Дефиниция (Свободни променливи)

Дефинираме функцията $\mathrm{FV} : \Lambda \to \mathcal{P}(V)$:

1. $\mathrm{FV}(x) := \{x\}$
2. $\mathrm{FV}(M_1 M_2) := \mathrm{FV}(M_1) \cup \mathrm{FV}(M_2)$
3. $\mathrm{FV}(\lambda_x N) := \mathrm{FV}(N) \setminus \{x\}$

Дефиниция (Затворени термове, комбинатори)

Терма $M \in \Lambda$ наричаме *затворен* или *комбинатор*, ако $\mathrm{FV}(M) = \emptyset$.

Свободни и свързани променливи

Дефиниция (Свободни променливи)

Дефинираме функцията $\mathrm{FV} : \Lambda \to \mathcal{P}(V)$:

1. $\mathrm{FV}(x) := \{x\}$
2. $\mathrm{FV}(M_1 M_2) := \mathrm{FV}(M_1) \cup \mathrm{FV}(M_2)$
3. $\mathrm{FV}(\lambda_x N) := \mathrm{FV}(N) \setminus \{x\}$

Дефиниция (Затворени термове, комбинатори)

Терма $M \in \Lambda$ наричаме *затворен* или *комбинатор*, ако $\mathrm{FV}(M) = \emptyset$.

Дефиниция (Свързани променливи)

Дефинираме функцията $\mathrm{BV} : \Lambda \to \mathcal{P}(V)$:

1. $\mathrm{BV}(x) := \emptyset$
2. $\mathrm{BV}(M_1 M_2) := \mathrm{BV}(M_1) \cup \mathrm{BV}(M_2)$
3. $\mathrm{BV}(\lambda_x N) := \mathrm{BV}(N) \cup \{x\}$

Наивна субституция

Дефиниция (Наивна субституция)

Нека $M, N \in \Lambda$, $x \in V$. Дефинираме субституцията на x с N в M, която ще отбелязваме с $M[x \rightsquigarrow N]$.

Наивна субституция

Дефиниция (Наивна субституция)

Нека $M, N \in \Lambda$, $x \in V$. Дефинираме субституцията на x с N в M, която ще отбелязваме с $M[x \rightsquigarrow N]$.

1. $x[x \rightsquigarrow N] := N$

Наивна субституция

Дефиниция (Наивна субституция)

Нека $M, N \in \Lambda$, $x \in V$. Дефинираме субституцията на x с N в M, която ще отбелязваме с $M[x \rightsquigarrow N]$.

1. $x[x \rightsquigarrow N] := N$
2. $y[x \rightsquigarrow N] := y$ за $y \not\equiv x$

Наивна субституция

Дефиниция (Наивна субституция)

Нека $M, N \in \Lambda$, $x \in V$. Дефинираме субституцията на x с N в M, която ще отбелязваме с $M[x \rightsquigarrow N]$.

1. $x[x \rightsquigarrow N] := N$
2. $y[x \rightsquigarrow N] := y$ за $y \not\equiv x$
3. $(M_1 M_2)[x \rightsquigarrow N] := (M_1[x \rightsquigarrow N])(M_2[x \rightsquigarrow N])$

Наивна субституция

Дефиниция (Наивна субституция)

Нека $M, N \in \Lambda$, $x \in V$. Дефинираме субституцията на x с N в M, която ще отбелязваме с $M[x \rightsquigarrow N]$.

1. $x[x \rightsquigarrow N] := N$
2. $y[x \rightsquigarrow N] := y$ за $y \not\equiv x$
3. $(M_1 M_2)[x \rightsquigarrow N] := (M_1[x \rightsquigarrow N])(M_2[x \rightsquigarrow N])$
4. $(\lambda_y P)[x \rightsquigarrow N] := \lambda_y(P[x \rightsquigarrow N])$

Проблеми при наивната субституция (1)

- Да разгледаме $(\lambda_{x,y}\, y)[y \rightsquigarrow x]$

Проблеми при наивната субституция (1)

- Да разгледаме $(\lambda_{x,y} y)[y \rightsquigarrow x] \equiv \lambda_{x,y} x$!

Проблеми при наивната субституция (1)

- Да разгледаме $(\lambda_{x,y} y)[y \rightsquigarrow x] \equiv \lambda_{x,y} x$!
- **Проблем:** термът си сменя смисъла след субституция!

Проблеми при наивната субституция (1)

- Да разгледаме $(\lambda_{x,y} y)[y \rightsquigarrow x] \equiv \lambda_{x,y} x$!
- **Проблем:** термът си сменя смисъла след субституция!
- Това явление се нарича “субституция на свързани променливи”

Проблеми при наивната субституция (1)

- Да разгледаме $(\lambda_{x,y} y)[y \rightsquigarrow x] \equiv \lambda_{x,y} x$!
- **Проблем:** термът си сменя смисъла след субституция!
- Това явление се нарича “субституция на свързани променливи”
- Как да го избегнем?

Проблеми при наивната субституция (1)

- Да разгледаме $(\lambda_{x,y} y)[y \rightsquigarrow x] \equiv \lambda_{x,y} x$!
- **Проблем:** термът си сменя смисъла след субституция!
- Това явление се нарича “субституция на свързани променливи”
- Как да го избегнем?
- **Идея №1:** да забраним субституция на свързани променливи

Наивна субституция

Дефиниция (Наивна субституция)

Нека $M, N \in \Lambda$, $x \in V$. Дефинираме субституцията на x с N в M, която ще отбелязваме с $M[x \rightsquigarrow N]$.

1. $x[x \rightsquigarrow N] := N$
2. $y[x \rightsquigarrow N] := y$ за $y \not\equiv x$
3. $(M_1 M_2)[x \rightsquigarrow N] := (M_1[x \rightsquigarrow N])(M_2[x \rightsquigarrow N])$
4. $(\lambda_x P)[x \rightsquigarrow N] := \lambda_x P$
5. $(\lambda_y P)[x \rightsquigarrow N] := \lambda_y (P[x \rightsquigarrow N])$, ако $x \not\equiv y$

Проблеми при наивната субституция (2)

- Да разгледаме $\lambda_x y[y \rightsquigarrow x]$

Проблеми при наивната субституция (2)

- Да разгледаме $\lambda_x y[y \rightsquigarrow x] \equiv \lambda_x x$!

Проблеми при наивната субституция (2)

- Да разгледаме $\lambda_x y[y \rightsquigarrow x] \equiv \lambda_x x$!
- **Проблем:** термът отново си сменя смисъла след субституция!

Проблеми при наивната субституция (2)

- Да разгледаме $\lambda_x y[y \rightsquigarrow x] \equiv \lambda_x x$!
- **Проблем:** термът отново си сменя смисъла след субституция!
- Това явление се нарича “прихващане на свободни променливи”

Проблеми при наивната субституция (2)

- Да разгледаме $\lambda_x y[y \rightsquigarrow x] \equiv \lambda_x x$!
- **Проблем:** термът отново си сменя смисъла след субституция!
- Това явление се нарича “прихващане на свободни променливи”
- Как да го избегнем?

Проблеми при наивната субституция (2)

- Да разгледаме $\lambda_x y[y \rightsquigarrow x] \equiv \lambda_x x$!
- **Проблем:** термът отново си сменя смисъла след субституция!
- Това явление се нарича “прихващане на свободни променливи”
- Как да го избегнем?
- **Идея №2:** да забраним прихващането

Частична субституция

Дефиниция (Частична субституция)

Нека $M, N \in \Lambda$, $x \in V$. Дефинираме субституцията на x с N в M, която ще отбелязваме с $M[x \hookrightarrow N]$.

1. $x[x \hookrightarrow N] := N$
2. $y[x \hookrightarrow N] := y$ за $y \not\equiv x$
3. $(M_1 M_2)[x \hookrightarrow N] := (M_1[x \hookrightarrow N])(M_2[x \hookrightarrow N])$
4. $(\lambda_x P)[x \hookrightarrow N] := \lambda_x P$
5. $(\lambda_y P)[x \hookrightarrow N] := \lambda_y(P[x \hookrightarrow N])$, ако $y \not\equiv x$ и $x \notin \mathrm{FV}(P)$ или $y \notin \mathrm{FV}(N)$

Проблем: функцията е частична!

Частична субституция

Дефиниция (Частична субституция)

Нека $M, N \in \Lambda$, $x \in V$. Дефинираме субституцията на x с N в M, която ще отбелязваме с $M[x \hookrightarrow N]$.

1. $x[x \hookrightarrow N] := N$
2. $y[x \hookrightarrow N] := y$ за $y \not\equiv x$
3. $(M_1 M_2)[x \hookrightarrow N] := (M_1[x \hookrightarrow N])(M_2[x \hookrightarrow N])$
4. $(\lambda_x P)[x \hookrightarrow N] := \lambda_x P$
5. $(\lambda_y P)[x \hookrightarrow N] := \lambda_y(P[x \hookrightarrow N])$, ако $y \not\equiv x$ и $x \notin \mathrm{FV}(P)$ или $y \notin \mathrm{FV}(N)$

Проблем: функцията е частична!

Идея №3: да добавим външна проверка за коректност

Коректна наивна субституция

Дефиниция (Коректна наивна субституция)

Казваме, че наивната субституция $M[x \rightsquigarrow N]$ е *коректна*, ако $\mathrm{FV}(N) \cap \mathrm{BV}(M) = \emptyset$.

Коректна наивна субституция

Дефиниция (Коректна наивна субституция)

Казваме, че наивната субституция $M[x \rightsquigarrow N]$ е *коректна*, ако $\mathrm{FV}(N) \cap \mathrm{BV}(M) = \emptyset$.

Задача

Да се покаже, че:

1. *ако $M[x \rightsquigarrow N]$ е коректна, то $M[x \hookrightarrow N]$ е дефинирана и $M[x \hookrightarrow N] \equiv M[x \rightsquigarrow N]$*
2. *има случай, в който $M[x \hookrightarrow N]$ е дефинирана, но $M[x \rightsquigarrow N]$ не е коректна*

Коректна наивна субституция

Дефиниция (Коректна наивна субституция)

Казваме, че наивната субституция $M[x \rightsquigarrow N]$ е *коректна*, ако $\mathrm{FV}(N) \cap \mathrm{BV}(M) = \emptyset$.

Задача

Да се покаже, че:

1. *ако $M[x \rightsquigarrow N]$ е коректна, то $M[x \hookrightarrow N]$ е дефинирана и $M[x \hookrightarrow N] \equiv M[x \rightsquigarrow N]$*
2. *има случай, в който $M[x \hookrightarrow N]$ е дефинирана, но $M[x \rightsquigarrow N]$ не е коректна*

Проблем: проверката за коректност е прекалено груба!

Коректна наивна субституция

Дефиниция (Коректна наивна субституция)

Казваме, че наивната субституция $M[x \rightsquigarrow N]$ е *коректна*, ако $\mathrm{FV}(N) \cap \mathrm{BV}(M) = \emptyset$.

Задача

Да се покаже, че:

1. *ако $M[x \rightsquigarrow N]$ е коректна, то $M[x \hookrightarrow N]$ е дефинирана и $M[x \hookrightarrow N] \equiv M[x \rightsquigarrow N]$*
2. *има случай, в който $M[x \hookrightarrow N]$ е дефинирана, но $M[x \rightsquigarrow N]$ не е коректна*

Проблем: проверката за коректност е прекалено груба!

Идея №4: да преименуваме свързаните променливи при нужда

Преименуване на свързаните променливи

Дефиниция (Субституция на Curry)

1. $x[x \mapsto N] := N$
2. $y[x \mapsto N] := y$ за $y \not\equiv x$
3. $(M_1 M_2)[x \mapsto N] := (M_1[x \mapsto N])(M_2[x \mapsto N])$
4. $(\lambda_x P)[x \mapsto N] := \lambda_x P$
5. $(\lambda_y P)[x \mapsto N] := \lambda_y (P[x \mapsto N])$, ако $y \not\equiv x$ и $x \notin \mathrm{FV}(P)$ или $y \notin \mathrm{FV}(N)$
6. $(\lambda_y P)[x \mapsto N] := \lambda_z (P[y \mapsto z][x \mapsto N])$ във всички останали случаи, където $z \notin \mathrm{FV}(P) \cup \mathrm{FV}(N)$

Преименуване на свързаните променливи

Дефиниция (Субституция на Curry)

1. $x[x \mapsto N] := N$
2. $y[x \mapsto N] := y$ за $y \not\equiv x$
3. $(M_1 M_2)[x \mapsto N] := (M_1[x \mapsto N])(M_2[x \mapsto N])$
4. $(\lambda_x P)[x \mapsto N] := \lambda_x P$
5. $(\lambda_y P)[x \mapsto N] := \lambda_y (P[x \mapsto N])$, ако $y \not\equiv x$ и $x \notin \mathrm{FV}(P)$ или $y \notin \mathrm{FV}(N)$
6. $(\lambda_y P)[x \mapsto N] := \lambda_z (P[y \mapsto z][x \mapsto N])$ във всички останали случаи, където $z \notin \mathrm{FV}(P) \cup \mathrm{FV}(N)$

Задача

Да се докаже, че горната дефиниция е коректна.

Преименуване на свързаните променливи

Дефиниция (Субституция на Curry)

1. $x[x \mapsto N] := N$
2. $y[x \mapsto N] := y$ за $y \not\equiv x$
3. $(M_1 M_2)[x \mapsto N] := (M_1[x \mapsto N])(M_2[x \mapsto N])$
4. $(\lambda_x P)[x \mapsto N] := \lambda_x P$
5. $(\lambda_y P)[x \mapsto N] := \lambda_y (P[x \mapsto N])$, ако $y \not\equiv x$ и $x \notin \mathrm{FV}(P)$ или $y \notin \mathrm{FV}(N)$
6. $(\lambda_y P)[x \mapsto N] := \lambda_z (P[y \mapsto z][x \mapsto N])$ във всички останали случаи, където $z \notin \mathrm{FV}(P) \cup \mathrm{FV}(N)$

Проблем: Горната субституция е релация, а не функция!

Преименуване на свързаните променливи

Дефиниция (Субституция на Curry)

1. $x[x \mapsto N] := N$
2. $y[x \mapsto N] := y$ за $y \not\equiv x$
3. $(M_1 M_2)[x \mapsto N] := (M_1[x \mapsto N])(M_2[x \mapsto N])$
4. $(\lambda_x P)[x \mapsto N] := \lambda_x P$
5. $(\lambda_y P)[x \mapsto N] := \lambda_y (P[x \mapsto N])$, ако $y \not\equiv x$ и $x \notin \mathrm{FV}(P)$ или $y \notin \mathrm{FV}(N)$
6. $(\lambda_y P)[x \mapsto N] := \lambda_z (P[y \mapsto z][x \mapsto N])$ във всички останали случаи, където $z \notin \mathrm{FV}(P) \cup \mathrm{FV}(N)$

Проблем: Горната субституция е релация, а не функция!

Идея №5: Да разглеждаме λ-термовете с точност до преименуване на свързаните променливи.

α-еквивалентност

Задача

Да се дефинира индуктивно релация на еквивалентност $\overset{\alpha}{=}$, така че $M \overset{\alpha}{=} N$ точно тогава, когато M може да се получи от N чрез подходящо преименуване на някои от свързаните си променливи.

α-еквивалентност

Задача

Да се дефинира индуктивно релация на еквивалентност $\overset{\alpha}{=}$, така че $M \overset{\alpha}{=} N$ точно тогава, когато M може да се получи от N чрез подходящо преименуване на някои от свързаните си променливи.

Нека разгледаме фактор-множеството $\Lambda_{/\overset{\alpha}{=}}$

α-еквивалентност

Задача

Да се дефинира индуктивно релация на еквивалентност $\overset{\alpha}{=}$, така че $M \overset{\alpha}{=} N$ точно тогава, когато M може да се получи от N чрез подходящо преименуване на някои от свързаните си променливи.

Нека разгледаме фактор-множеството $\Lambda_{/\overset{\alpha}{=}}$,
т.е. вместо отделни λ-термове разглеждаме класове на еквивалентност от λ-термове относно релацията $\overset{\alpha}{=}$.

α-еквивалентност

Задача

Да се дефинира индуктивно релация на еквивалентност $\overset{\alpha}{=}$, така че $M \overset{\alpha}{=} N$ точно тогава, когато M може да се получи от N чрез подходящо преименуване на някои от свързаните си променливи.

Нека разгледаме фактор-множеството $\Lambda_{/\overset{\alpha}{=}}$,
т.е. вместо отделни λ-термове разглеждаме класове на еквивалентност от λ-термове относно релацията $\overset{\alpha}{=}$.

Задача

Да се докаже, че субституцията на Curry е функция над фактор-множеството $\Lambda_{/\overset{\alpha}{=}}$

α-еквивалентност

Задача

Да се дефинира индуктивно релация на еквивалентност $\overset{\alpha}{=}$, така че $M \overset{\alpha}{=} N$ точно тогава, когато M може да се получи от N чрез подходящо преименуване на някои от свързаните си променливи.

Нека разгледаме фактор-множеството $\Lambda_{/\overset{\alpha}{=}}$,
т.е. вместо отделни λ-термове разглеждаме класове на еквивалентност от λ-термове относно релацията $\overset{\alpha}{=}$.

Задача

Да се докаже, че субституцията на Curry е функция над фактор-множеството $\Lambda_{/\overset{\alpha}{=}}$,
т.е. ако $M \overset{\alpha}{=} M' \in \Lambda$, $N \overset{\alpha}{=} N' \in \Lambda$, то $M[x \mapsto N] \overset{\alpha}{=} M'[x \mapsto N']$.

α-еквивалентност (2)

Задача

Да се покаже, че винаги можем да намерим представител на класа на еквивалентност на даден терм, за който наивната субституция да е коректна

α-еквивалентност (2)

Задача

Да се покаже, че винаги можем да намерим представител на класа на еквивалентност на даден терм, за който наивната субституция да е коректна,
т.е. за всяко $M \in \Lambda$ можем да намерим $M' \stackrel{\alpha}{=} M$, така че $M'[x \rightsquigarrow N]$ да е коректна.

α-еквивалентност (2)

Задача

Да се покаже, че винаги можем да намерим представител на класа на еквивалентност на даден терм, за който наивната субституция да е коректна,
т.е. за всяко $M \in \Lambda$ можем да намерим $M' \overset{\alpha}{=} M$, така че $M'[x \rightsquigarrow N]$ да е коректна.

Задача

Да се покаже, че операцията за частична субституция може да се разглежда като тотална с точност до релацията $\overset{\alpha}{=}$

α-еквивалентност (2)

Задача

Да се покаже, че винаги можем да намерим представител на класа на еквивалентност на даден терм, за който наивната субституция да е коректна,
т.е. за всяко $M \in \Lambda$ можем да намерим $M' \overset{\alpha}{=} M$, така че $M'[x \rightsquigarrow N]$ да е коректна.

Задача

Да се покаже, че операцията за частична субституция може да се разглежда като тотална с точност до релацията $\overset{\alpha}{=}$, т.е.

1. *За всяко $M \in \Lambda$, съществува $M' \overset{\alpha}{=} M$, така че $M'[x \hookrightarrow N]$ е дефинирана.*
2. *Ако $M \overset{\alpha}{=} M' \in \Lambda$, $N \overset{\alpha}{=} N' \in \Lambda$ и $M[x \hookrightarrow N]$ и $M'[x \hookrightarrow N']$ са дефинирани едновременно, то $M[x \hookrightarrow N] \overset{\alpha}{=} M'[x \hookrightarrow N']$.*

Конвенции на Barendregt

След като се уверихме, че има коректно и формално решение за избягване на прихващането на променливи, за удобство ще въведем следните две конвенции, предложени от Barendregt:

Конвенции на Barendregt

След като се уверихме, че има коректно и формално решение за избягване на прихващането на променливи, за удобство ще въведем следните две конвенции, предложени от Barendregt:

- Отъждествяваме синтактично термове, които са α-еквивалентни, т.е. считаме че релацията $\overset{\alpha}{=}$ е "вградена" в синтактичното равенство $\equiv$.

Конвенции на Barendregt

След като се уверихме, че има коректно и формално решение за избягване на прихващането на променливи, за удобство ще въведем следните две конвенции, предложени от Barendregt:

- Отъждествяваме синтактично термове, които са α-еквивалентни, т.е. считаме че релацията $\overset{\alpha}{=}$ е "вградена" в синтактичното равенство $\equiv$.
- Ако в даден момент разглеждаме някакво множество от термове, ще считаме че сме избрали такива представители, че нито една от свързаните променливи няма да съвпада с нито една от свободните.

Конвенции на Barendregt

След като се уверихме, че има коректно и формално решение за избягване на прихващането на променливи, за удобство ще въведем следните две конвенции, предложени от Barendregt:

- Отъждествяваме синтактично термове, които са α-еквивалентни, т.е. считаме че релацията $\overset{\alpha}{=}$ е "вградена" в синтактичното равенство $\equiv$.
- Ако в даден момент разглеждаме някакво множество от термове, ще считаме че сме избрали такива представители, че нито една от свързаните променливи няма да съвпада с нито една от свободните.

Задача

Да се направи програмна реализация на операцията субституция, която при нужда преименува свързаните променливи по подходящ начин при прилагане, за да осигури коректност.

Свойства на субституцията

Лема 1

Нека $M, N \in \Lambda$ *и* $x \notin \mathrm{FV}(M)$. *Тогава* $M[x \mapsto N] \equiv M$.

Свойства на субституцията

Лема 1

Нека $M, N \in \Lambda$ *и* $x \notin \mathrm{FV}(M)$. *Тогава* $M[x \mapsto N] \equiv M$.

Лема 2 (за субституцията)

Нека $M, N, P \in \Lambda$, *а* $x \not\equiv y \in V$ *и* $x \notin \mathrm{FV}(P)$. *Тогава*
$M[x \mapsto N][y \mapsto P] \equiv M[y \mapsto P][x \mapsto N[y \mapsto P]]$.