λ–rachunek

Inspiracja

Materiały do ćwiczeń będą oparte w dużej mierze na materiałach dr. Sławomira Bakalarskiego (do tego samego kursu).

Te materiały, a wykład

Celem tych materiałów jest bardziej przekazanie intuicji do tego, jak działa rachunek λ, niż zdefiniowanie wszystkiego od podstaw. Duża część pojawiających się definicji pokrywa się jednak z wykładem (oraz jest na nim w większej lub mniejszej części oparta).

Zmienne

Zaczynamy od przeliczalnego zbioru *zmiennych* V. Jak zobaczymy, w praktyce w każdym wyrażeniu będzie się znajdowało tylko skończenie wiele zmiennych. Równocześnie ważnym elementem przekształcania wyrażeń w wyrażenia równoważne jest "zamiana nazw zmiennych", warto dostrzec, że nieskończoność zbioru zmiennych jest w istocie uproszczeniem (ponieważ chcąc zmienić nazwę zmiennej x na jakąś, która w wyrażeniu nigdzie nie występuje nie musimy się przejmować, czy "nie skończył się alfabet").

Termy

Termy są słowami nad alfabetem $V \cup \{(,), ., \lambda\}$ zdefiniowanymi rekurencyjnie przez reguły:

- Pojedyncze wystąpienie zmiennej jest termem.
- Jeśli t_1, t_2 to termy, $(t_1 t_2)$ jest termem (*aplikacja*).
- Jeśli t jest termem, a x zmienną, to $(\lambda x.t)$ jest termem (*abstrakcja*).

Zakładając, że wszystkie litery alfabetu łacińskiego są elementami V, przykładami termów są: x, y, (xy), $((z(xy))x)$, $(\lambda x.x)$, $(\lambda x.y)$, $(\lambda x.(xy))$, $(\lambda x.((z(xy))x))$

Konwencje

W praktyce pomija się w zapisie dużą część nawiasów oraz pomija się część znaków λ w określonych sytuacjach.

- "Najbardziej zewnętrzne" nawiasy są pomijane.
- Jeśli nastąpiło kilka abstrakcji pod rząd, zapisujemy "lewe strony" razem przed pojedynczą kropką i piszemy tylko pierwszy znak λ, np. zamiast $(\lambda x.(\lambda y.(xy))$ możemy napisać $(\lambda xy.(xy))$.
- Jeśli nastąpiło kilka aplikacji pod rząd, to zapisując je bez nawiasów mamy na myśli wykonywanie ich od lewej, tzn. zamiast $((AB)C)$ możemy napisać ABC, ale $(A(BC))$ możemy uprościć już tylko do $A(BC)$.
- W przypadku abstrakcji pomijamy najbardziej zewnętrzne nawiasy po prawej stronie kropki. Inaczej mówiąc, po wystąpieniu kropki wyobrażamy sobie, że nawias zaczyna się zaraz po niej, a zamyka się "jak najbardziej po prawej", np. $\lambda xy.xy(xy)$ jest uproszczonym zapisem $\lambda xy.(xy(xy))$.

Konwencje c.d.

Warto pamiętać, że konwencje te mają być tylko uproszczeniem. Czasem lepiej jest "zostawić" nadmiarowe nawiasy, jeśli zwiększa to czytelność. Jest to też tylko konwencja zapisu, czyli powinniśmy mieć przez cały czas świadomość, jak wygląda wersja "w pełni nawiasowana" naszego wyrażenia.

Aplikowanie funkcji do zmiennych

O termach postaci $\lambda xy.F$ możemy generalnie myśleć jako o funkcji F, która przyjmuje argumenty x i y, natomiast F jest "ciałem funkcji" (w którym mogą, i często występują, zmienne x i y). Aplikacja odpowiada wtedy niejako temu, jak działa to w Haskellu, tzn. $(\lambda xy.F)\ A\ B$ odpowiada wywołaniu funkcji polegającym na zastąpieniu "każdego" (niekoniecznie! o tym za chwilę) wystąpienia zmiennej x w F przez A oraz zmiennej y w F przez B. Analogia działa też przy "wywołaniu częściowym", tzn. podobnie, jak w Haskellu funkcję dwóch zmiennych wywołaną na jednej zmiennej możemy traktować jak funkcję jednej zmiennej, tak $(\lambda xy.F)\ A$ odpowiada $\lambda y.F'$, gdzie F' powstaje z F przez zastąpienie wystąpień x przez A. Możemy myśleć o $((\lambda xy.F)\ A)$ jako o funkcji jednej zmiennej, której zaaplikowanie do B doprowadzi nas do "tego samego" co aplikacja $(\lambda xy.F)\ A\ B$.

Zmienne wolne i związane

Pojęcie *zmiennej wolnej* dla danego termu M również jest definiowane rekurencyjnie, następującymi regułami:

- Jeśli M jest pojedynczą zmienną, to jest ona wolna.
- Jeśli M jest postaci PQ, to zmienne wolne M są sumą zbiorów zmiennych wolnych P i zbioru zmiennych wolnych Q.
- Jeśli M jest postaci $\lambda x.M'$, to zbiór zmiennych wolnych M jest zbiorem zmiennych wolnych M' z wyrzuconym x ($\lambda x.$ "wiąże" wolne wystąpienia x w M'.

Zbiór zmiennych wolnych dla termu M oznaczamy przez $FV(M)$. Zmienna występująca w M jest *związana*, jeśli nie jest wolna. Term M jest *domknięty*, jeśli $FV(M) = \varnothing$.
Przykład: w termie $\lambda xy.yz$ zmienna y jest związana, a z wolna. Zmienną x najlepiej tu opisać jako wiążącą. W termie $x(\lambda x.x)$ pierwsze wystąpienie zmiennej x jest wolne (nie jest związane przez żadne "$\lambda x.$" przed nim).

Redukcje

Naszym celem jest teraz wprowadzenie mechanizmów, które umożliwiają nam symulowanie wywoływania funkcji. Wyobraźmy sobie, że chcemy zaaplikować term $\lambda xy.F$ (gdzie x, y występują w F) do termu A. Chcielibyśmy zastąpić w F wszystkie wystąpienia x przez A. Co jednak w przypadku, gdy w A występuje y? Nic w konstrukcji termów tego nie zabrania, z drugiej strony powinniśmy intuicyjnie czuć, że $\lambda ab.ab$ oraz $\lambda cd.cd$ powinny reprezentować te same funkcje, więc wynik aplikacji tej funkcji do termu "b" nie powinien dawać dwóch różnych wyników w zależności od "przyjętego reprezentanta" naszej funkcji.

α–redukcja

α–redukcja odpowiada właśnie zamianie nazw zmiennych, którą chcemy przeprowadzić w celu uniknięcia konfliktu (bądź zwiększenia przejrzystości). Równoważność termów z dokładnością do α–redukcji będziemy zapisywać przez $=_\alpha$ (nie jest te jedyny możliwy zapis, można spotkać też np. $\rightarrow_\alpha$). Intuicyjnie chodzi o fakt, że term $\lambda x.M$, gdzie w M nie występuje y jest α–równoważny termowi $\lambda y.M'$, gdzie M' powstaje z M przez zastąpienie **wszystkich** wystąpień x przez y. Formalnie jest to najmniejsza w sensie zawierania relacja, która jednocześnie respektuje warunek powyżej oraz spełnia warunek bycia kongruencją (mówiąc nieformalnie kongruencja jest relacją równoważności zachowującą się "dobrze" ze względu na działania). Intuicyjnie odpowiada to faktowi, że możemy wykonywać α–redukcje tylko na "kawałku" termu.
Przykłady: $\lambda ab.ab =_\alpha \lambda cd.cd$, ale również $x(\lambda x.x) =_\alpha x(\lambda y.y)$.

β–redukcja

β–redukcja odpowiada niejako wywoływaniu funkcji, chcemy wszystkie wystąpienia zmiennej zastąpić przez jakiś term.
Będziemy zapisywać β–redukcję przez $\rightarrow_\beta$ (można spotkać też np. $=_\beta$, ale może być to o tyle mylące, że β–redukcja nie jest symetryczna, a po znaku $=$ spodziewamy się zwykle symetrii).
Powiemy, że term $(\lambda x.M)N$ β–redukuje się do termu M', który powstaje z M przez zastąpienie wszystkich wolnych wystąpień x przez N, ale w taki sposób, że wszystkie zmienne, które były wolne w N mają być wolne w M'.
Na przykład $(\lambda xy.xy)z \rightarrow_\beta (\lambda y.zy)$, ale $(\lambda xy.xy)y$ nie β–redukuje się do $(\lambda y.yy)$. W pierwszym przypadku zmienna z, do której zaaplikowaliśmy funkcję jest wolna przed i po aplikacji, w drugim już by tak nie było. Takich konfliktów będziemy unikać przez α–redukcje.

β–redukcja wraz z α–redukcją

Przykład użycia α–redukcji w celu uniknięcia konfliktów:
$(\lambda xy.x)y =_\alpha (\lambda xz.x)y \rightarrow_\beta \lambda z.y =_\alpha \lambda x.y$

Stałe i podstawowe funkcje

W rachunku λ możemy reprezentować stałe (np. liczby naturale, prawda, fałsz) oraz funkcje na nich. Jest to poniekąd podobna sytuacja do definiowania liczb naturalnych na teorii mnogości przy pomocy zbiorów. Po tym, jak już przejdziemy formalności związane z tym, czy np. funkcja następnika, a później dodawania czy mnożenia zachowuje się "dobrze", przestaje nam być aż tak potrzebna świadomość, że liczby to tak na prawdę pewne określone zbiory, korzystamy z uproszczonego zapisu i tego, jak działają podstawowe funkcje. W rachunku lambda sytuacja jest podobna, przyjmujemy, że pewne termy reprezentują sobą stałe, a później udowadniamy, że kolejne termy reprezentują sobą działania na tych stałych. Zaczniemy od stałych logicznych (prawda, fałsz) oraz funkcjach na nich jako prostszych od analogonów dla liczb naturalnych.

Prawda, fałsz, funkcje logiczne

Definiujemy:

- $T = \lambda xy.x$,
- $F = \lambda xy.y$,
- $NOT = \lambda x.(x\ F\ T)$,
- $AND = \lambda xy.xyx$,
- $OR = \lambda xy.xxy$.

Tutaj T odpowiada prawdzie, F fałszowi, NOT negacji, AND koniunkcji oraz OR alternatywie. Zauważmy, że T jako funkcja dwóch zmiennych odpowiada też rzutowaniu na pierwszą współrzędną, a F na drugą. Może to być istotne w sprawdzaniu, że funkcje NOT, AND oraz OR zachowują się "tak, jak chcemy", jednak kiedy już to sprawdzimy i będziemy operowali na zasadzie redukowania $OR\ (AND\ T\ F)\ T$ do T, wiedza, jaką dokładnie funkcję reprezentuje sobą F czy T nie będzie aż tak istotna.

Przykład redukcji na stałych logicznych

Mamy:
$NOT\ T = (\lambda x.x\ F\ T)T \rightarrow_\beta T\ F\ T = (\lambda xy.x)\ F\ T \rightarrow_\beta F$.
Oczywiście już na początku moglibyśmy "rozpisać" wszystkie wystąpienia F i T (używając α–redukcji w celu uniknięcia konfliktów):
$NOT\ T = (\lambda x.x\ F\ T)T \rightarrow_\beta T\ F\ T =$
$(\lambda x.x\ (\lambda ab.b)\ (\lambda cd.c))(\lambda ef.e) \rightarrow_\beta (\lambda ef.e)\ (\lambda ab.b)\ (\lambda cd.c) \rightarrow_\beta$
$(\lambda f.(\lambda ab.b))\ (\lambda cd.c) \rightarrow_\beta (\lambda ab.b) =_\alpha (\lambda xy.y) = F$.

Bibliografia (do materiałów dr. S. Bakalarskiego, na których się opierałem)

- Paul Hudak.
  A Brief and Informal Introduction to the Lambda Calculus.
  url:
  http://www.cs.yale.edu/homes/hudak/CS201S08/lambda.pdf.
- Bruce J. MacLennan.
  Functional programming practice and theory. Addison-Wesley, 1990.
- Wazniak. Paradygmaty programowania. url:
  http://wazniak.mimuw.edu.pl/index.php?title=
  Paradygmaty_programowania.
- Wikipedia. Fixed-point combinator. url: https:
  //en.wikipedia.org/wiki/Fixed-point_combinator#
  Fixed_point_combinators_in_lambda_calculus.
- Wikipedia. Rachunek lambda. url:
  https://pl.wikipedia.org/wiki/Rachunek_lambda.