Obliczymy dla przykładu wartość $(\mathbf{Y}\ F)\ 3$:

$$\begin{aligned}(\mathbf{Y}\ F)\ 3 &= F\ (\mathbf{Y}\ F)\ 3\\ &= (3 \leqslant 1)\ 1\ (3 * ((\mathbf{Y}\ F)\ 2))\\ &= 3 * (F\ (\mathbf{Y}\ F)\ 2)\\ &= 3 * ((2 \leqslant 1)\ 1\ (2 * ((\mathbf{Y}\ F)\ 1)))\\ &= 3 * (2 * (F\ (\mathbf{Y}\ F)\ 1))\\ &= 3 * (2 * ((1 \leqslant 1)\ 1\ ((\mathbf{Y}\ F)\ 0)))\\ &= 3 * 2 * 1 = 6 = 3!\end{aligned}$$

Uwaga: „nieskończone złożenie” zostało oczywiście sprytnie ukryte. W wyrażeniu na $\mathbf{Y}$ występują bowiem funkcje (x oraz $\lambda x.\ F\ (x\ x)$), które są swoimi własnymi argumentami. Nie stwarza to jednak problemu – zwróćmy uwagę, że już wcześniej wyszliśmy poza ścisłe, matematyczne znaczenie słowa „funkcja”, nie troszcząc się, na przykład, o zdefiniowanie dziedziny: argumentem funkcji $\lambda x.x$ może być równie dobrze liczba 25, funkcja $\lambda y.2y$, jak i jakikolwiek inny obiekt. W każdym przypadku wyrażenie $\lambda x.x$ ma sens i jego wartością jest dostarczony argument.

Efekt ten uzyskamy za pomocą specjalnego – „magicznego” lambda-wyrażenia, nazywanego operatorem punktu stałego:

$$\mathbf{Y} = \lambda F.(\lambda x.F\ (x\ x))\ (\lambda x.F\ (x\ x))$$

Zauważmy, że

$$\begin{aligned}\mathbf{Y}\ F &= (\lambda x.\ F\ (x\ x))\ (\lambda x.\ F\ (x\ x))\\ &= F\ ((\lambda x.\ F\ (x\ x))\ (\lambda x.\ F\ (x\ x)))\\ &= F\ (\mathbf{Y}\ F)\\ &= F\ (F\ (\mathbf{Y}\ F))\\ &= F\ (F\ (F\ (\mathbf{Y}\ F)))\\ &\ldots\end{aligned}$$

Czyli mamy to, czego szukaliśmy: $\mathbf{Y}\ F$ jest „nieskończenie wiele razy złożonym F”, więc szukaną funkcją silnia.

⋆ ⋆ ⋆

Podsumowując, zdefiniowaliśmy formalny system, w którym za pomocą zaledwie kilku konstrukcji możemy programować. Jednocześnie daje on poręczną notację służącą do definiowania i posługiwania się funkcjami. A dzięki swojej prostocie pozwala stosunkowo prosto wnioskować o własnościach tego języka programowania.

Typy i tautologie

Rachunek lambda w swych założeniach miał być systemem formalnym leżącym u podstaw całej matematyki. Dopiero gdy ten program zawiódł, wykorzystano powstały formalizm do zdefiniowania funkcji obliczalnych. Później okazało się, że po odpowiednim wzbogaceniu może służyć także swojemu pierwotnemu celowi. Przyjrzymy się teraz związkom rachunku lambda z logiką i rozumowaniem.

Wiemy już, że lambda-wyrażenie postaci $\lambda x.M$ reprezentuje pewną funkcję $f(x)$. Lambda-wyrażenie M należy wtedy interpretować jako definicję tej funkcji, czyli po prostu algorytm, który ją oblicza. W językach programowania zwykle nadajemy argumentom typy oraz określamy typ wyniku. W języku C napisalibyśmy np.

```
int f(int x) { return 5*x; }
```

definiując funkcję $f : \text{int} \to \text{int}$ (dla nieprogramistów: $\text{int} = \mathbb{Z}$).

W takim przypadku będziemy mówili, że *typem* funkcji f jest $\text{int} \to \text{int}$. Interesują nas tutaj tylko takie wyrażenia, w których wszystkie zmienne są związane z pewną lambdą, czyli na przykład $\lambda x.\lambda y.y\ x$, ale nie $\lambda y.y\ x$. Ponieważ lambda-wyrażenia są funkcjami, to powinniśmy móc im także nadać pewne typy.

Weźmy dla przykładu wyrażenie $\lambda x.x$, czyli funkcję identycznościową. Argument x w tym przypadku może być dowolnego typu. Nazwijmy ten typ α. Typ wyniku musi być taki sam jak typ zmiennej x, czyli też α. W takim razie typem wyrażenia $\lambda x.x$ jest po prostu $\alpha \to \alpha$.

Rozważmy nieco trudniejszy przykład $\lambda y.(\lambda x.x)$. Niech y będzie pewnego typu β, a x typu α. Wiemy już, że $\lambda x.x$ ma typ $\alpha \to \alpha$, więc $\lambda y.(\lambda x.x)$ będzie typu $\beta \to (\alpha \to \alpha)$.

Weźmy teraz wyrażenie $\lambda x.(\lambda y.y\ x)$. Jeśli x jest typu α, to y musi być funkcją, która przyjmuje argument typu α. Nie wiemy, jakiego typu jest wartość zwracana przez funkcję y, więc nazwiemy ten typ β. W takim razie y będzie typu $\alpha \to \beta$. W ten sposób możemy nadać wyrażeniu $\lambda x.(\lambda y.y\ x)$ typ

$$\alpha \to ((\alpha \to \beta) \to \beta).$$

Jako ćwiczenie Czytelnik może sprawdzić, że typem wyrażenia $\lambda x.\lambda y.\lambda z.(x\ z)\ (y\ z)$ jest

$$(\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma)).$$

Nie każdemu lambda-wyrażeniu da się nadać typ. Na przykład wyrażenie $\lambda x.x\ x$ nie ma żadnego typu.

Zauważmy, że jeśli zinterpretujemy „$\to$” jako znak implikacji, to nasze typy staną się formułami logiki zdaniowej (czyli takiej, w której nie ma kwantyfikatorów $\forall$ i $\exists$). **Okazuje się, że wszystkie formuły, jakie mogą w ten sposób powstać, są tautologiami!** W szczególności nie istnieje lambda-wyrażenie np. typu $(\alpha \to \beta) \to \beta$, bo taka formuła nie jest tautologią. Lambda-wyrażenie danego typu τ możemy zatem traktować jako dowód prawdziwości formuły τ. Daje nam to ciekawy sposób dowodzenia, że dana formuła τ jest tautologią. Wystarczy znaleźć lambda-wyrażenie M typu τ. Co ciekawe, dla formuł zawierających koniunkcję $\wedge$, alternatywę $\vee$ oraz kwantyfikatory $\forall$ i $\exists$ też istnieją „rachunki lambda” o takiej własności, że każda formuła, która jest typem jakiegoś lambda-wyrażenia, jest też tautologią.

Sławomir KOLASIŃSKI