Pełna inferencja typów z indukcją własności programów dla systemów GADT z arytmetyką

AUTOR ŁUKASZ STAFINIAK

Instytut Informatyki Uniwersytetu Wrocławskiego

Web: `www.ii.uni.wroc.pl/~lukstafi`

Inferencja dla systemu typów Hindleya-Milnera (HM) (leżącego u podstaw języków z rodziny ML) redukuje się do problemu unifikacji; gdy dodamy inferencję typów funkcji polimorficznie rekurencyjnych – system typów Milnera-Mycrofta (MM) – inferencja redukuje się do problemu semi-unifikacji. Rozszerzamy teraz HM do systemu HM(X): do schematów typów dodajemy więzy w postaci koniunkcji formuł atomowych, które mogą być równościami na typach (w HM(X) redundantne) lub należeć do dziedziny więzów X. Kolejnym krokiem jest wprowadzenie uogólnionych typów algebraicznych. Gdy chcemy zachować inferencję = unifikacji, ograniczamy schematy typów konstruktorów wartości do postaci $\forall\vec{\alpha}.\vec{\tau}\to\varepsilon(\vec{\alpha})$, gdzie ε jest konstruktorem typu algebraicznego. HMG(X) nie nakłada tego typu ograniczeń, w szczególności konstruktory mogą mieć dodatkowe więzy: $\forall\vec{\alpha}[D].\vec{\tau}_1\to\varepsilon(\vec{\tau}_2)$; dzięki więzom równoważnie mamy $\forall\vec{\alpha}\vec{\beta}[D].\vec{\tau}\to\varepsilon(\vec{\alpha})$ (biorąc $D\wedge\vec{\tau}_2\doteq\vec{\alpha}$ zamiast D). Więzy „przeszkadzają" gdy budujemy wartości, ale „pomagają" gdy rozbieramy wartości:

$$\langle\Gamma\vdash\text{case}\,e_0\,\text{of}\,\{K_1\Rightarrow e_1|...|K_n\Rightarrow e_n\}\colon\tau\rangle\longmapsto\langle\Gamma\vdash e_0\colon\gamma\rangle\wedge_i\forall\vec{\beta}_i.D_i\Rightarrow(\langle\Gamma\vdash e_i\colon\delta_i\rangle\wedge\delta_i\doteq\vec{\tau}_i\to\varepsilon(\vec{\alpha}))$$

gdzie $K_i::\forall\vec{\alpha}\vec{\beta}_i[D].\vec{\tau}_i\to\varepsilon(\vec{\alpha})$.

Postawiłem sobie zadanie inferencji typów dla systemu MMG(arith), z programami zupełnie pozbawionymi annotacji typami. W więzach oprócz równań na drzewach typów mam arytmetykę liniową na liczbach naturalnych. Chcę zgadywać typy takie jak:

$$\begin{aligned}\text{eval} &:: \forall t.\text{Term}(t)\to t\\ \text{split} &:: \forall m,n,k[k=m+n].\text{List}(k)\to(\text{List}(m),\text{List}(n))\\ \text{merge} &:: \forall m,n.(\text{List}(m),\text{List}(n))\to\text{List}(m+n)\\ \text{filter} &:: \forall n,k[n\leqslant k].\text{List}(k)\to\text{List}(n)\\ \text{mergesort} &:: \forall n.\text{List}(n)\to\text{EList}(n)\end{aligned}$$

gdzie m.in.:

$$\begin{aligned}\text{ONil} &:: \text{OList}(0,0)\\ \text{OCons} &:: \forall n,a,b[b\leqslant a].(\text{Nat}(a),\text{OList}(n,b))\to\text{OList}(n+1,a)\\ \text{Ex} &:: \forall n,a.\text{OList}(n,a)\to\text{EList}(n)\end{aligned}$$

Implementuję program inferujący takie typy dla języka programowania posiadającego inferencję rodzajów w definicjach typów, zagnieżdżone wzorce w „pattern matchingu"...

System, który implementuję, nie ma (jeszcze?) eleganckiej charakterystyki formalnej. Jest w nim element przeszukiwania kombinatorycznego ograniczonego przez rozmiar rozpatrywanych formuł. Opiera się na kilku pomysłach:

- uogólnieniu algorytmu Hengleina dla semiunifikacji, do sytuacji, gdy całe więzy mogą być w zakresie podstawień semiunifikacyjnych (wywołanie funkcji polimorficznej „instancjuje" również więz tej funkcji)
- "niezawodnym" algorytmie unifikacji-generalizacji (zastosowanym do typów poszczególnych gałęzi dopasowania wzorca)
- indukcji formuły charakteryzującej rekurencyjną definicję przez przypadki poprzez:
  - wygenerowanie zbioru „wszystkich" formuł wywodliwych z wszystkich gałęzi bazowych definicji
  - iteracyjne zawężanie tego zbioru przez sprawdzanie wywodliwości z gałęzi rekurencyjnych, gdzie w każdym kroku zakładamy (dodajemy do przesłanek gałęzi) odpowiednie instancje formuł z poprzedniego kroku.