Notatka o typach w języku Plait

Poniższa notatka stanowi materiał opcjonalny – znajomość opisanych treści nie będzie sprawdzana.

System typów języka Plait można opisać w sposób analogiczny do systemu dedukcji naturalnej znanego z „Logiki dla informatyków". Reguły typowania zapisane w ten sposób mogą pomóc zrozumieć, w jaki sposób język Plait wyznacza typy wyrażeń. Poniżej zapisany jest uproszczony zestaw takich reguł.

Dla jednoznaczności zapisu wprowadzimy pewne oznaczenia:

- Wyrażenia języka Plait (np. `(+ 2 3)`) będziemy oznaczać literą e, natomiast zmienne występujące w wyrażeniach – literą x.
- Typy języka Plait (np. `(Listof String)`) będziemy oznaczać literą t, natomiast zmienne typowe literą poprzedzoną apostrofem – np. $'a$.
- Literały typów bazowych (np. $0, 1$ typu `Number`, lub $\#t, \#f$ typu `Boolean`) będziemy zapisywać literą l z indeksem dolnym – np. $l_{\texttt{Integer}}$ dla literału typu Integer.
- Środowiska typów, opisujące typy zmiennych (a w zasadzie szablony typów), będziemy zapisywać literą Γ. Środowiska typów można traktować jak zbiory par złożonych z nazwy zmiennej i jej szablonu typu, na przykład $x : \forall' a_1 \ldots {}' a_n.t$. W sytuacji, gdy szablon typu nie zawiera żadnej kwantyfikowanej zmiennej, kwantyfikator będziemy pomijać, pisząc po prostu $x : t$.

Najprostszą regułą typowania jest reguła dla **zmiennych**. Mówi ona, że typem wyrażenia będącego zmienną jest *instancja* schematu typów tej zmiennej, tzn. wynik podstawienia wybranych typów za zmienne typowe kwantyfikowane w schemacie.

$$\frac{}{\Gamma, x : \forall' a_1 \ldots {}' a_n.t \vdash x : t[t_1/'a_1] \ldots [t_n/'a_n]}$$

Również prosta jest reguła typowania dla **literałów**. Każdy literał ma właściwy sobie typ prosty – np. `Number` dla liczb, `Boolean` dla wartości boolowskich, `Symbol` dla symboli.

$$\frac{}{\Gamma \vdash l_t : t}$$

Wyrażenia warunkowe wymagają, aby warunek był typu `Boolean`, a pozostałe podwyrażenia były tych samych typów.

$$\frac{\Gamma \vdash e : \texttt{Boolean} \qquad \Gamma \vdash e_1 : t \qquad \Gamma \vdash e_2 : t}{\Gamma \vdash \texttt{(if } e\ e_1\ e_2\texttt{)} : t}$$

Nową zmienną do środowiska typów może wprowadzić lambda-wyrażenie lub wyrażenie definiujące (np. let-wyrażenie). Reguła typowania dla **lambda-wyrażeń** wprowadza do środowiska szablony typów bez kwantyfikatora ogólnego (z pustym zbiorem kwantyfikowanych zmiennych):

$$\frac{\Gamma, x_1 : t_1, \ldots, x_n : t_n \vdash e : t}{\Gamma \vdash \texttt{(lambda (}x_1 \ldots x_n\texttt{) } e\texttt{)} : \texttt{(}t_1 \ldots t_n\ \texttt{->}\ t\texttt{)}}$$

Wyrażenia typu funkcyjnego (ze strzałką) można **aplikować**. Typy wyrażeń aplikowanych do funkcji muszą się zgadzać z typami argumentów (po lewej od strzałki):

$$\frac{\Gamma \vdash e : \texttt{(}t_1 \ldots t_n\ \texttt{->}\ t\texttt{)} \qquad \Gamma \vdash e_1 : t_1 \qquad \ldots \qquad \Gamma \vdash e_n : t_n}{\Gamma \vdash \texttt{(}e\ e_1 \ldots e_n\texttt{)} : t}$$

Nowe zmienne mogą wprowadzić też **let-wyrażenia**. W przeciwieństwie do lambda-wyrażeń, let-wyrażenia automatycznie kwantyfikują świeże (nie występujące wcześniej w środowisku typów) zmienne typowe. Zbiór wolnych zmiennych typowych w zadanym typie będziemy oznaczać $\mathrm{FV}(t)$. W poniższej regule, T_k oznacza wszystkie zmienne wolne z t_k, które nie występują w Γ (czyli $T_k = \mathrm{FV}(t_k) \setminus \mathrm{FV}(\Gamma)$).

$$\frac{\Gamma \vdash e_1 : t_1 \qquad \ldots \qquad \Gamma \vdash e_n : t_n \qquad \Gamma, x_1 : \forall T_1.t_1, \ldots, x_n : \forall T_n.t_n \vdash e : t}{\Gamma \vdash \texttt{(let ([}x_1\ e_1\texttt{]} \ldots \texttt{[}x_n\ e_n\texttt{]) } e\texttt{)} : t}$$

Wyrażenia **letrec** tym się różnią od zwykłych let-wyrażeń, że ciała definicji mogą odwoływać się do siebie samych. Opisuje to poniższa reguła. *Uwaga! To nie jest rzeczywista reguła typowania języka Plait. Język Plait dopuszcza tak zwaną rekursję polimorficzną, której nie można otypować regułą poniżej. Na zajęciach nie będziemy używać rekursji polimorficznej, dzięki czemu wykorzystanie uproszczonej reguły nie będzie powodować problemów.*

$$\frac{\begin{array}{c}\Gamma, x_1 : t_1, \ldots, x_n : t_n \vdash e_1 : t_1 \qquad \ldots \\ \Gamma, x_1 : t_1, \ldots, x_n : t_n \vdash e_n : t_n \qquad \Gamma, x_1 : \forall T_1.t_1, \ldots, x_n : \forall T_n.t_n \vdash e : t\end{array}}{\Gamma \vdash \texttt{(letrec ([}x_1\ e_1\texttt{]}\ldots\texttt{[}x_n\ e_n\texttt{]) }e\texttt{)} : t}$$