Zadanie domowe 4

W niniejszym zadaniu domowym zajmiemy się implementacją systemu typów w języku TML (Typed ML), który nie jest niczym innym tylko rachunkiem λ z typami prostymi bez typu void.

Abstrakcyjny syntaks TML jest podany poniżej. W porównaniu z abstrakcyjnym syntaksem rachunku λ z typami prostymi podanym na wykładzie, zawiera dodatkowy typ podstawowy int dla liczb całkowitych, ale pomija typ void, który w praktycznych zastosowaniach jest niepotrzebny. $\hat{n}$ oznacza liczbę całkowitą n, natomiast plus i minus są operacjami arytmetycznymi na liczbach całkowitych; eq sprawdza, czy dane dwie liczby są równe.

$$\begin{array}{rrcl}
\mathsf{typ} & A & ::= & \mathsf{bool}|\mathsf{int}|A \to A|A \times A|\mathsf{unit}|A + A \\
\mathsf{wyrażenie} & e & ::= & x|\lambda x : A.e|e\ e|(e,e)|\mathsf{fst}e|\mathsf{snd}e|() | \\
 & & & \mathsf{inl}_A e|\mathsf{inl}_A e|\mathsf{case}\ e\ \mathsf{of}\ \mathsf{inl}x.e|\mathsf{inr}x.e|\mathsf{fix}x : A.e| \\
 & & & \mathsf{true}|\mathsf{false}|\mathsf{if}\ e\ \mathsf{then}\ e\ \mathsf{else}\ e|\hat{n}|\mathsf{plus}|\mathsf{minus}|\mathsf{eq} \\
\mathsf{kontekst\ typujący} & \Gamma & ::= & \cdot|\Gamma, x : A
\end{array}$$

Konkretny syntaks TML jest podany poniżej. Wszystkie wyrażenia po prawej stronie poniższej definicji podane są w takiej samej kolejności, jak ich odpowiedniki w powyższym abstrakcyjnym syntaksie:

$$\begin{array}{rrcl}
\mathtt{typ} & A & ::= & \mathtt{bool}|\mathtt{int}|A->A|A*A|\mathtt{unit}|A+A|(A) \\
\mathtt{wyrażenie} & e & ::= & x|\mathtt{fn}x : A => e|e\ e|(e,e)|\mathtt{fst}e|\mathtt{snd}e|() | \\
 & & & \mathtt{inl}(A)e|\mathtt{inl}(A)e|\mathtt{case}\ e\ \mathtt{of}\ \mathtt{inl}x => e|\mathtt{inr}x => e|\mathtt{fix}x : A => e| \\
 & & & \mathtt{true}|\mathtt{false}|\mathtt{if}\ e\ \mathtt{then}\ e\ \mathtt{else}\ e|\hat{n}|+|-|= \\
 & & & \mathtt{let}x : A = e\ \mathtt{in}\ e'|\mathtt{let\ rec}\ x : A = e\ \mathtt{in}\ e'|(e) \\
\mathtt{liczba\ całkowita} & \hat{n} & ::= & \ldots|\mathtt{\tilde{}2}|\mathtt{\tilde{}1}|\mathtt{0}|\mathtt{1}|\mathtt{2}|\ldots
\end{array}$$

Dodajemy następujące dwa słodziki syntaktyczne:

$$\begin{array}{rcl}
\mathtt{let}x : A = e\ \mathtt{in}\ e' & \text{dla} & (\mathtt{fn}\ x : A => e')e \\
\mathtt{let\ rec}\ x : A = e\ \mathtt{in}\ e' & \text{dla} & (\mathtt{fn}x : A => e')(\mathtt{fix}x : A.e)
\end{array}$$

(A) jest takie samo jak A i służy do uproszczenia prawej łączności $\mathtt{->}$. Na przykład, $A_1 -> A_2 -> A_3$ jest równoważne z $A_1 -> (A_2 -> A_3)$, które jest różne od $(A_1 -> A_2) -> A_3$. Podobnie (e) jest takie samo jak e i służy uproszczeniu lewej łączności aplikacji. Na przykład, $e_1 e_2 e_3$ jest równe $(e_1 e_2)e_3$, które jest różne od $e_1(e_2 e_3)$.

Celem niniejszego zadania jest zaimplementowanie poniższego systemu typów TML:

$$\frac{x : A \in \Gamma}{\Gamma \vdash x : A}\mathsf{Var} \qquad \frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash \lambda x : A.e : A \to B} \to \mathsf{I} \qquad \frac{\Gamma \vdash e : A \to B\ \Gamma \vdash e' : A}{\Gamma \vdash e\ e' : B} \to \mathsf{E}$$

$$\frac{\Gamma \vdash e_1 : A_1\ \Gamma \vdash e_2 : A_2}{\Gamma \vdash (e_1, e_2) : A_1 \times A_2} \times \mathsf{I} \qquad \frac{\Gamma \vdash e : A_1 \times A_2}{\Gamma \vdash \mathsf{fst}e : A_1} \times \mathsf{E}_1 \qquad \frac{\Gamma \vdash e : A_1 \times A_2}{\Gamma \vdash \mathsf{snd}e : A_2} \times \mathsf{E}_2 \qquad \frac{}{\Gamma \vdash () : \mathsf{unit}}\mathsf{Unit}$$

$$\frac{\Gamma \vdash e : A_1}{\Gamma \vdash \mathsf{inl}_{A_2} e : A_1 + A_2} + \mathsf{I}_\mathsf{L} \qquad \frac{\Gamma \vdash e : A_2}{\Gamma \vdash \mathsf{inr}_{A_1} e : A_1 + A_2} + \mathsf{I}_\mathsf{R}$$

$$\frac{\Gamma \vdash e : A_1 + A_2\ \Gamma, x_1 : A_1 \vdash e_1 : C\ \Gamma, x_2 : A_2 \vdash e_2 : C}{\Gamma \vdash \mathsf{case}\ e\ \mathsf{of}\ \mathsf{inl}x_1.e_1|\mathsf{inr}x_2.e_2 : C} + \mathsf{E}$$

$$\frac{\Gamma, x : A \vdash e : A}{\Gamma \vdash \mathsf{fix}x : A.e : A}\mathsf{Fix}$$

$$\frac{}{\Gamma \vdash \mathsf{true} : \mathsf{bool}}\mathsf{True} \qquad \frac{}{\Gamma \vdash \mathsf{false} : \mathsf{bool}}\mathsf{False} \qquad \frac{\Gamma \vdash e : \mathsf{bool}\ \Gamma \vdash e_1 : A\ \Gamma \vdash e_2 : A}{\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : A}\mathsf{If}$$

$$\frac{}{\Gamma \vdash \hat{n} : \mathsf{int}}\mathsf{Int}$$

$$\frac{}{\Gamma \vdash \mathsf{plus} : \mathsf{int} \times \mathsf{int} \to \mathsf{int}}\,\mathsf{Plus} \qquad \frac{}{\Gamma \vdash \mathsf{minus} : \mathsf{int} \times \mathsf{int} \to \mathsf{int}}\,\mathsf{Minus} \qquad \frac{}{\Gamma \vdash \mathsf{eq} : \mathsf{int} \times \mathsf{int} \to \mathsf{bool}}\,\mathsf{Eq}$$

1. Instrukcja programowania

Ściągnijcie ze strony kursu archiwum .zip z plikami do zadania:

`http://www.math.us.edu.pl/~pgladki/teaching/2013-2014/log_lab4.zip`

a następnie rozpakujcie je do Waszego katalogu roboczego. Zobaczycie całe mnóstwo plików w głównym katalogu oraz podkatalogi `parsing` i `util`, do których nie musicie zaglądać (chyba, że z ciekawości).

Na początek zapoznajcie się ze strukturą `Tml` w `tml.sml` o sygnaturze `TML`.

```
structure Tml : TML =
struct
  type var = string
  datatype tp =
    ...
  datatype exp =
    ...
end
```

Datatypy `tp` i `exp` odpowiadają kategoriom syntaktycznym `typ` oraz `wyrażenie`, odpowiednio. Przeczytajcie komentarze w pliku i upewnijcie się, że rozumiecie do czego służą poszczególne konstruktory danych.

Następnie przyjrzyjcie się `typing-sig.sml` oraz `typing.sml`. Celem niniejszego zadania jest zaimplementowanie funkcji `typing` w strukturze `Typing`:

```
val typing : context -> Tml.exp -> Tml.tp
```

Innymi słowy `typing` bierze kontekst Γ o typie `Typing.context` i wyrażenie e o typie `Tml.exp` i zwraca typ A taki, że $\Gamma \vdash e : A$ lub zwraca wyjątek `TypeError`, jeśli nie istnieje typ A taki, że $\Gamma \vdash e : A$.

Zauważcie, że sami musicie zadecydować, jak ma wyglądać definicja typu w `Typing.context`. W dołączonym pliku `Typing.context` jest zdefiniowany jako `unit`, ale powinniście zamienić go na odpowiedni typ dla kontekstów typujących. Gdy już zdecydujecie się na definicję `Typing.context`, będziecie musieli zaimplementować funkcję `createEmptyContext`, która zwraca pusty kontekst:

```
val createEmptyContext : unit -> context
```

Następnie spróbujcie zaimplementować regułę Var aby upewnić się, że będziecie w stanie odzyskać poszczególne elementy z kontekstu typującego.

Będziecie musieli się zastanowić jak zaimplementować $\Gamma, x : A$, które jest wymagane przez reguły $\to\mathsf{I}$, $+\mathsf{E}$ oraz Fix. Jeżeli będziecie chcieli jako niezmiennik to, że zmienne w kontekście typującym są parami rozłączne, będziecie potrzebowali α-konwersji dla tych reguł. Okazuje się jednak, że można tego uniknąć – zastanówcie się w jaki sposób!

Jeżeli będziecie chcieli użyć `(Tml.var*Tml.tp) list` dla `Typing.context`, zastanówcie się, czy wiecie co robicie ;) `(Tml.var*Tml.tp) list` to niewątpliwie dobry kandydat na definicję `Typing.context`, ale spróbujcie zastanowić się nad lepszym i bardziej eleganckim rozwiązaniem.

Celem przetestowania Waszego kodu będziecie chcieli wykorzystać funkcje w strukturze `Loop` w `loop.sml` (nie musicie szczegółowo analizować plików `loop-sig.sml` oraz `loop.sml`). Zanim zaczniecie, upewnijcie się, że zainstalowana przez Was wersja SML-a pozwala na włączenie trybu wartościowania w SML-u dla leniwych (używanej przez parser). W linii komend wpiszcie po prostu `Control.lazysml := true;` celem włączenia trybu wartościowania dla leniwych; w przeciwnym razie otrzymacie potem wiadomość taką jak `syntax error: replacing ID with LAZY`:

```
- Control.lazysml := true;
[autoloading]
  ...
[autoloading done]
val it = () : unit
-
```

Następnie skompilujcie pliki źródłowe za pomocą Compile Managera:

```
- CM.make "sources.cm";
...
[New bindings added.]
val it = true : bool
-
```

Następnie otwórzcie strukturę `Loop`:

```
- open Loop;
opening Loop
  ...
-
```

Teraz możecie testować Waszą funkcję `typing` wpisując `loop (step show);` celem przejścia w tryb TML. Następnie wpiszcie wyrażenie TML zakończone średnikiem ;. Jeśli wyrażeniu można przypisać prawidłowy typ, zostanie on wyświetlony wraz z rezultatem parsowania, w przeciwnym razie zostanie zgłoszony błąd.

```
- loop (step show);
Tml> fn x:int => x;
fn x : int => x: int -> int
Tml>x;
x has no type.
```

Pamiętajcie o tym, że jeżeli dokonacie zmian w Waszej funkcji `typing`, przed ponownym użyciem będziecie musieli uruchomić Compile Managera i otworzyć ponownie strukturę `Loop`. W przeciwnym razie będziecie dalej używać starej wersji funkcji `typing`.

Gdy uznacie, że zrobiliście już wszystko, co potrafiliście zrobić, wyślijcie plik typing.sml emailem do prowadzącego. Termin składania zadania mija 1 stycznia 2014 roku. Proszę pamiętać, aby w polu subject Waszego maila umieścić tag [aghlog]. Pod żadnym pozorem nie kompresujcie wysyłanych plików, nie zmieniajcie ich nazw, nie wysyłajcie całego katalogu lab3 ani nie róbcie żadnej z nieskończonego ciągu nieprzewidywalnych rzeczy, które moglibyście zrobić – po prostu wyślijcie maila z dołączonym plikiem i to wszystko.