Zadanie domowe 3

Po wstępnej rozgrzewce z programowania w SML-u i kilku prostych ćwiczeniach dotyczących indukcji, pora na trochę bardziej ambitny projekt. Celem dzisiejszego zadania domowego będzie zaimplementowanie semantyki operacyjnej wywołania według wartości rachunku λ. Głównym zamierzeniem nie będzie wszakże sama implementacja, ale raczej nabycie umiejętności **interpretowania reguł wnioskowania jako algorytmów,** co jest z kolei bezwzględnie konieczną umiejętnością potrzebną w implementowaniu języków programowania. Innymi słowy mając dany system reguł wnioskowania będziemy chcieli wyekstrahować z nich algorytm odpowiadający regułom – właśnie w tym ma nam pomóc niniejsze zadanie.

Rozważmy następujące trzy reguły redukcyjne dla osądu redukcyjnego $e \mapsto e'$:

$$\frac{e_1 \mapsto e_1'}{e_1 e_2 \mapsto e_1' e_2} Lam \qquad \frac{e_2 \mapsto e_2'}{(\lambda x.e) e_2 \mapsto (\lambda x.e) e_2'} Arg \qquad \frac{}{(\lambda x.e) v \mapsto [v/x]e} App$$

Reguła *Lam* powiada, że jeśli e_1 redukuje się do e_1' to wówczas $e_1 e_2$ redukuje się do $e_1' e_2$. Podobnie możemy wytłumaczyć sobie pozostałe dwie reguły. Taka dosłowna interpretacja pozwala nam odpowiedzieć na pytanie czy dana **redukcja** jest poprawna. Koncepcyjnie dane wejściowe do danego problemu to osąd redukcyjny $e \mapsto e'$, zaś odpowiedź brzmi "tak" wraz z drzewem wnioskowania uzasadniającym redukcję, albo "nie ". Tym samym dane wejściowe do problemu składają się z dwóch wyrażeń: wyrażenia e oraz drugiego wyrażenia e', do którego e może lub nie może się zredukować.

Wszelako podczas implementowania semantyki operacyjnej będziemy przede wszystkim zainteresowani odpowiedzią na pytanie jak zredukować dane **wyrażenie.** W tym wypadku dane wejściowe do problemu to pewne wyrażenie e, zaś odpowiedź to albo inne wyrażenie e' wraz z drzewem dowodowym osądu $e \mapsto w'$, lub "nie", co oznacza, że nie istnieje wyrażenie do którego e mogłoby się zredukować. Tym samym musimy interpretować reguły redukcyjne nie dosłownie, ale **algorytmicznie**. W tym szczególnym przypadku, musimy zinterpretować reguły redukcyjne od wniosków do przesłanek, a więc w kierunku od dołu do góry.

Zinterpretujmy zatem reguły redukcyjne algorytmicznie. Jako że danymi wejściowymi do problemu jest wyrażenie, zaś danymi wyjściowymi jest inne wyrażenie lub "nie", zacznijmy od wprowadzenia funkcji `step` o następującej specyfikacji:

- `step` pobiera wyrażenie e.
- `step` e zwraca nowe wyrażenie e' jeśli e redukuje się do e'.
- `step` e zgłasza wyjątek jeśli nie istnieje wyrażenie e' do którego e się redukuje.

Spróbujmy przepisać regułę *Lam* z użyciem funkcji `step`. Intuicyjnie regułę *Lam* możemy interpretować jak następuje:

(1) Rozważmy przypadek, gdy `step` pobiera jako argument wyrażenie $e_1 e_2$.
(2) `step` wywołuje rekursywnie e_1, ponieważ chce zdecydować do jakiego wyrażenia redukuje się e_1, o ile w ogóle do jakiegoś.
(3) Jeśli `step` e_1 zwraca e_1', to wówczas zwracamy $e_1' e_2$.
(4) Jeśli `step` e_1 zgłasza wyjątek, przekazujemy go dalej.

W podobny sposób możemy zinterpretować pozostałe reguły. Celem tej części zadania domowego jest zaimplementowanie funkcji `step` do implementowania reguł redukcyjnych.

Podczas implementacji funkcji `step` będziecie zapewne chcieli wyekstrahować funkcje z definicji indukcyjnych

- $FV(e)$ dla zmiennych wolnych w wyrażeniu e,
- $[e'/x]e$ dla podstawiania e' za x w e,
- $e \equiv_\alpha e'$ dla α-równoważności pomiędzy e oraz e'.

W odróżnieniu od poprzednich zadań domowych, w których podaliśmy *explicite* typy wszystkich funkcji, jakie trzeba było zaimplementować, w niniejszym zadaniu nie podamy specyfikacji funkcji do zaimplementowania – podamy tylko ich definicje indukcyjne, które można znaleźć w notatkach do wykładu. Interesuje nas tylko poprawność działania funkcji `step` i nic ponadto. Przykładowo, jeśli uznacie, że w implementacji funkcji `step` nie jest Wam potrzebna funkcja α-równoważności, możecie ją pominąć.

Główna motywacja skłaniająca do tego diabolicznego zabiegu to zamiar nauczenia Was najważniejszych zasad w projektowaniu oprogramowania, a mianowicie zwrócenie Waszej uwagi w kierunku designu i specyfikacji. Jak już wkrótce sami się przekonacie, większość pracy inżyniera informatyka polega na wymyśleniu "co zaimplementować", nie zaś "jak zaimplementować" – to ostatnie jest zadaniem dla programistów. Tym samym najważniejsza rada mająca pomóc Wam w dzisiejszym zadaniu jest następująca: **solidnie się zastanówcie zanim zaczniecie cokolwiek wklepywać do komputera**. W istocie nie musicie nawet uruchamiać komputerów, zanim "na kartce" nie rozpiszecie sobie dokładnie tego, jakie funkcje trzeba zaimplementować, o jakich typach, jakich niezmiennikach itd.

1. Instrukcja programowania

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

`http://www.math.us.edu.pl/~pgladki/teaching/2013-2014/log_lab3.zip`

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

Na początek zapoznajcie się ze strukturą `Uml` w `uml.sml` o sygnaturze `UML`. `UML` jest skrótem od `Untyped ML`, a Waszym zadaniem będzie implementowanie interpretera UML-a, jak będziemy w tym zadaniu nazywać rachunek λ (ściślej: beztypowy (*untyped*) rachunek λ, czyli ten, który na razie poznaliśmy na wykładach, w odróżnieniu od rachunku λ z typami prostymi, o którym będziemy się uczyć już za chwilę).

```
structure Uml : UML =
struct
  type var = string
  datatype exp =
    Var of var
  | Lam of var * exp
  | App of exp * exp
end
```

Datatyp `exp` odpowiada kategorii syntaktycznej `expression` w notatkach z wykładu:

- `Var` x oznacza zmienną x jako wyrażenie w UML.
- `Lam (`x`,` e`)` oznacza λ-abstrakcję $\lambda x.e$ w UML.
- `App (`e_1`,` e_2`)` oznacza aplikację $e_1 e_2$ w UML.

Następnie przyjrzyjcie się `eval-sig.sml` oraz `eval.sml`. Celem niniejszego zadania jest zaimplementowanie funkcji `step`:

```
(*redukcja jesdnego kroku, zgłasza Stuck gdy redukcja jest niemożliwa*)
val set : Uml.exp -> Uml.exp
```

Innymi słowy `step` bierze wyrażenie e typu `Uml.exp` i zwraca nowe wyrażenie e' do którego redukuje się e; jeśli nie istnieje takie wyrażenie e', funkcja zgłasza wyjątek `Stuck`. Redukcja wykorzystuje strategię wywołania według wartości (a nie wywołania według nazwy).

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]
[library $smlnj/compiler/current.cm is stable]
[library $smlnj/compiler/x86.cm is stable]
[library $smlnj/viscomp/core.cm is stable]
...
[library $SMLNJ-MLRISC/Control.cm is stable]
[library $controls-lib.cm(=$SMLNJ-LIB/Controls)/controls-lib.cm is stable]
[library $smlnj/smlnj-lib/controls-lib.cm is stable]
[autoloading done]
val it = () : unit
-
```

Następnie skompilujcie pliki źródłowe za pomocą Compile Managera:

```
- CM.make "sources.cm";
...
[compiling (sources.cm):loop.sml]
[code: 3613, data: 73, env: 166 bytes]
[New bindings added.]
val it = true : bool
-
```

Prawdopodobnie napotkacie na kilka ostrzeżeń, które wszakże będziecie mogli zignorować. Następnie otwórzcie strukturę `Loop`:

```
- open Loop;
opening Loop
  type action = Uml.exp -> unit
  val show : action
  val eval : action -> action
  val step : action -> action
  val wait : action -> action
  val loop : action -> unit
  val loopFile : string -> action -> unit
-
```

Teraz możecie testować Waszą funkcję `loop` na dwa sposoby. Jeżeli wpiszecie `loop (step (wait show));` przy linii komand, to będziecie mogli wpisać wyrażenie UML zakończone średnikiem `;`. Wkrótce podamy szczegółowy syntaks UML-a.

```
- loop (step (wait show));
Uml> (lam x. x) (lam y. y);
((lam x. x) (lam y. y))
Press return:
...
```

Za każdym razem, gdy wciśniecie klawisz enter, pojawi się zredukowane wyrażenie. Możecie też wykorzystać wyrażenie UML przechowywane w osobnym pliku. Do dyspozycji macie trzy pliki: `nat.ml`, `rec.uml` oraz `fib.uml`.

```
- loopFile "nat.uml" (step (wait show));
...
```

Jeśli chcecie zobaczyć cały ciąg redukcji bez wciskania klawisza enter, użyjcie `step show`:

```
- loopFile "nat.uml" (step show);
...
```

Jeśli chcecie pominąć wszystkie kroki pośrednie i zobaczyć tylko końcowy rezultat, użyjcie `eval show`:

```
- loopFile "nat.uml" (eval show);
...
```

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

```
- CM.make "sources.cm";
...
- open Loop;
opening Loop
...
```

2. Syntaks UML

Syntaks UML do złudzenia przypomina syntaks rachunku λ. Jedyna różnica polega na użyciu słowa kluczowego `lam` zamiast λ oraz słodzika syntaktycznego `let` $x - e$ `in` e' dla (`lam`$x.e'$)e.

$$\text{wyrażenie} \qquad e ::= x | \texttt{lam}x.e | ee | \texttt{let}x = e\texttt{in}e$$

Na przykład następujące wyrażenie UML wyznacza liczebnik Churcha dla liczby naturalnej 8 (do znalezienia w `nat.uml`):

```
let one = lam s. lam z. s z in
let add = lam x. lam y. lam s. lam z. y s (x s z) in
let two = add one one in
let four = add two two in
let eight = add four four in
eight
;
```

Gdy uznacie, że zrobiliście już wszystko, co potrafiliście zrobić, wyślijcie plik eval.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.