# Lista zagadnień nr 9

## Na zajęciach

**Ćwiczenie 1.**

Jakie domknięcie będzie wartością wyrażeń (przedstawionych w jakieś tam składni konkretnej) w języku z wykładu?

- `(let (x 5) (lambda (z) (let (y 5) (+ x y z))))`
- `(let (x 5) (lambda (x) (let (y 5) (+ x y))))`
- `((lambda (x) (lambda (y) (+ x y))) 10)`

**Ćwiczenie 2.**

Napisz w języku z wykładu z konstrukcją `letrec` funkcje `reverse`, `map` i `append`.

**Ćwiczenie 3.**

W kodzie przedstawionym na wykładzie domknięecie przechowuje ciało procedury i środowisko aktualne w momencie utworzenia procedury. Nie zawsze jednak trzeba przechowywać całe środowisko – wystarczy znać wartości zmiennych używanych przez procedurę. Zdefiniuj procedurę, która potrafi stworzyć takie okrojone środowisko i użyj jej w interpreterze.

**Ćwiczenie 4.**

Zdefiniuj procedurę sprawdzającą, czy dwa wyrażenia są $\alpha$-równoważne.

**Ćwiczenie 5.**

Alternatywnym sposobem na reprezentowanie składni abstrakcyjnej z wiązaniem zmiennych są tzw. *indeksy de Bruijna*. Jest to reprezentacja, która nie używa nazw na zmienne. Wystąpienie zmiennej związanej nie jest dane przez jej nazwę,

ale przez jej **indeks**, czyli liczbę wiązań zmiennych pomiędzy wystąpieniem, a wiążącym ją binderem. Na przykład wyrażenie dane przez Racketową składnię konkretną następująco

```
(lambda (x)
  (+ x
     (let ((y 4))
       (* x y))))
```

można wyrazić przy pomocy indeksów de Bruijna mniej więcej tak (zakładając, że `lambda` i `let` wiążą zawsze jedną zmienną):

```
(lambda
  (+ (index 0)
     (let 4
       (* (index 1) (index 0)))))
```

Zwróć uwagę, że zmienna `x` raz reprezentowana jest przez indeks 0 (bo między nią, a miejscem, gdzie jest wiązana, nie ma innych binderów), a raz przez indeks 1 (bo pomiędzy wystąpieniem związanym a wiążącym jest jeszcze `let`). Jeśli używamy indeksów de Bruijna, wyrażenia `(+ x x)` ma postać `(+ (index )`

Użycie indeksów de Bruijna do reprezentowania składni abstrakcyjnej ma kilka zalet. Po pierwsze, $\alpha$-równoważne wyrażenia są po prostu równe. Po drugie, środowiska można reprezentować zwyczajnie jako listę wartości. Rozważ program

```
(let ((x 4)) (+ x x))
```

Tak jak robiliśmy to do tej pory, po obliczeniu wartości wyrażenia `4`, obliczamy wyrażenie `(+ x x)` ze środowiskiem, w którym zmiennej `x` przypisano wartość 4. Teraz, skoro nie mamy nazw zmiennych, środowisko to po prostu lista wartości. Wyrażenie `(+ x x)` reprezentowane jest jako `(+ (index 0) (index 0))`, co oznacza, że dodajemy wartość zmiennej, znajdującej się na zerowej (licząc od zera) pozycji w środowisku.

- Zaproponuj reprezentację składni abstrakcyjnej wyrażeń, która używa indeksów de Bruijna.
- Zmodyfikuj interpeter tak, by działał na tej składni.
- Napisz procedurę, która konwertuje składnię z wykładu na składnię z indeksami de Bruijna.
- Napisz procedurę, która konwertuje składnię z indeksami de Bruijna na składnię z wykładu. Może przyda się infomracja o tym, jak wygenerować nowy symbol, zamieszczona w zad 8. z listy 8.

**Ćwiczenie 6.**

Funkcja Ackermanna $\text{Ack} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ zdefiniowana jest następującą zależnością rekurencyjną:

$$\begin{aligned}
&\text{Ack}(0,\, n) = n + 1 \\
&\text{Ack}(m,\, 0) = \text{Ack}(m - 1,\, 1) && \text{dla } m > 0 \\
&\text{Ack}(m,\, n) = \text{Ack}(m - 1,\, \text{Ack}(m,\, n - 1)) && \text{dla } m > 0 \text{ oraz } n > 0
\end{aligned}$$

Bardzo łatwo zdefiniować ją w Rackecie, przepisując bezpośrednio powyższą zależność jako funkcję rekurencyjną. Zdefiniuj funkcję Ackermanna w języku WHILE dla wyrażeń, które operują na liczbach, wartościach boolowskich, parach, listach i symbolach (ale nie funkcjach). Innymi słowy, dla języka WHILE zdefiniowanego w plikach `w10-expr.rkt` i `w10-while-interp.rkt` na SKOS-ie.

*Wskazówka:* Przypomnij sobie, jak implementuje się wywołania funkcji za pomocą stosu. W trzeciej klauzuli funkcji Ack chcemy policzyć najpierw $\text{Ack}(m,\, n-1)$, ale po uzyskaniu wyniku (np. $r$), chcemy powrócić do klauzuli i obliczyć $\text{Ack}(m-1, r)$. W czasie liczenia wartości $\text{Ack}(m,\, n-1)$ trzeba gdzieś zapamiętać, co będziemy robić po uzyskaniu wyniku. Gdzie? Najlepiej na stosie.

*Lepsza wskazówka:* Najpierw zaimplementuj w Rackecie **iteracyjną** wersję funkcji Ack. A potem po prostu przepisz iterację na pętlę `while`.