Zadanie na pracownię nr 7

Uwaga: Termin oddania rozwiązań tego zadania to sobota, 14 kwietnia 2018, godz 6:00.

Wyciąganie let-definicji na zewnątrz wyrażeń

W tym zadaniu rozważamy wyrażenia arytmetyczne z `let`-ami, zgodnie z udostępnionym na SKOS-ie szablonem. Zadaniem jest zaimplementować pewną transformację na wyrażeniach. Transformacja ta to wyciąganie let-definicji na zewnątrz wyrażeń. Dla przykładu, rozważmy następujące wyrażenie w naszym interpretownym języku wyrażeń:

```
(+ 10 (* (let (x 7) (+ x 2)) 2))
```

Okazuje się, że definicję `(x 7)` można równie dobrze zadeklarować na poziomie najbardziej zewnętrznym (z angielskiego zwanym toplevelem). Otrzymjemy wówczas równoważne wyrażenie:

```
(let (x 7) (+ 10 (* (+ x 2) 2)))
```

Podobne transformacje często wykonywane są przez kompilator w celu uzyskania kodu, który można skompilować do bardziej wydajnego rezultatu.

Trudność takiej transformacji wynika z faktu, że let-wyrażenia mogą być dowolnie zagnieżdżone, np.

```
(let (x (- 2 (let (z 3) z))) (+ x 2))
```

jest równoważne wyrażeniu

```
(let (z 3)
(let (x (- 2 z))
(+ x 2)))
```

ale **nie** jest równoważne wyrażeniu

```
(let (x (- 2 z))
(let (z 3)
(+ x 2)))
```

Dodatkowo, trzeba uważać na zmienne, które mogą się powtarzać w oryginalnym wyrażeniu, a z oczywistych względów nie mogą po transformacji. Dla przykładu, wyrażenie

```
(+ (let (x 5) x)
   (let (x 1) x))
```

nie jest równoważne wyrażeniu

```
(let (x 5)
(let (x 1)
  (+ x x)))
```

Trzeba odpowiednio przemianować zmienne, np.

```
(let (x 5)
(let (y 1)
  (+ x y)))
```

Zadanie polega na zdefiniowaniu jednoargumentowej procedury `let-lift`, która zmienia wyrażenia z dowolnymi let-definicjami w równoważne wyrażenia, które mają let-definicje tylko na zewnątrz, zgodnie z następującym predykatem:

```
(define (arith-expr? t)
  (or (const? t)
      (and (op? t)
           (andmap arith-expr? (op-args t)))
      (var? t)))

(define (let-lifted-expr? t)
  (or (and (let? t)
           (let-lifted-expr? (let-expr t))
           (arith-expr? (let-def-expr (let-def t))))
      (arith-expr? t)))
```

Świeże nazwy zmiennych

Jak widać w powyższych przykładach, nasza transformacja czasem wymaga generowania świeżych nazw zmiennych. Skąd wziąć świeże nazwy?

Jednym ze sposobów jest generowanie ich za pomocą licznika. Można to uzyskać np. dzięki następjącej procedurze:

```
(define (number->symbol i)
  (string->symbol (string-append "x" (number->string i))))
```

w której argument `i` jest liczbą.

Jak użyć takiego licznika? Standardowym wzorcem jest przyjęcie początkowej wartości licznika jako argumentu i zwracanie jego nowej razem z wynikiem procedury (które to rzeczy można połączyć przy użyciu pary). Np. poniższa procedura zmienia etykiety w liściach drzewa na kolejne liczby naturalne w porządku od lewej do prawej:

```
(define (rename-with-counter t i)
  (cond [(leaf? t) (cons (leaf-cons i) (+ 1 i))]
        [(node? t) (let ([r1 (rename-with-counter (left-subtree  t) i)]
                         [r2 (rename-with-counter (right-subtree t) (cdr r1))])
                     (cons (node-cons (car r1) (car r2)) (cdr r2)))]))

(define (rename t)
  (car (rename-with-counter t 0)))
```

Być może dobrym pomysłem jest zdefiniowanie własnego typu danych dla zwracanych wartości, by nie zakopać się w `car`-ach i `cdr`-ach.

Kiedy w ogóle przemianowywać zmienną w let-definicji na świeżą? Najprostszym, a zarazem poprawnym pomysłem jest robić to za każdym razem, gdy transformujemy let-wyrażenie.

Jak sprawić, że zmienne, które odnosiły się do starej nazwy, będą odnosić się do nowej nazwy? Najprościej użyć środowiska. Tym razem przypisywać może ono każdej zmiennej jej nową nazwę.

Uwaga: Rozwiązanie musi być czysto funkcyjne. Użycie wszelkich nieczystych konstrukcji Racketowych typu `gensym` jest **surowo zabronione**!

Wskazówka

Zadanie to można rozwiązać procedurą, która ma podobną strukturę jak ewaluator, czyli jest rekurencyjna względem struktury oryginalnego wyrażenia. Warto jednak rozważyć bardziej użyteczną reprezentację wyników pośrednich. Jedną z możliwości jest reprezentowanie wyrażeń jako listy let-definicji sparowanej z wyrażaniem arytmetycznym już bez let-definicji. Np. jeśli transforumujemy wyrażenie

```
(+ e1 e2)
```

i z transformacji `e1` dostaniemy listę definicji `ds1` i wyrażenie arytmetyczne `a1`, zaś z `e2` dostaniemy listę definicji `ds2` i wyrażenie arytmetyczne `a2`, to łatwo skonstruować z tego wewnętrzną reprezentację jako parę złożoną z listy `(append ds1 ds2)` i wyrażenia `(+ a1 a2)`.

Z takiej pary potem łatwo odrębną procedurą odbudować ostateczne wyrażenie z letami za zewnątrz.

Przesyłanie rozwiązań

Rozwiązanie należy porządnie przetestować i zamieścić testy w przełanym pliku.

Rozwiązanie należy przesłać jako plik o nazwie w formacie `nazwisko-imie.rkt`, jak zwykle bez spacji i polskich znaków. Rozwiązanie należy nadesłać w formie uzupełnienia szablonu dostępnego na SKOS-ie.