Lista zagadnień nr 2

Ćwiczenie 1.

W poniższych wyrażeniach zlokalizuj wolne i związane wystąpienia zmiennych. Które wystąpienia *wiążą* każde z wystąpień związanych?

```
x
```

```
(let ([x 3])
  (+ x y))
```

```
(let ([x 1]
      [y (+ x 2)])
  (+ x y))
```

```
(let ([x 1])
  (let ([y (+ x 2)])
    (* x y)))
```

```
(lambda (x y)
  (* x y z))
```

```
(let ([x 1])
  (lambda (y z)
    (* x y z)))
```

Ćwiczenie 2.

Złożenie funkcji f i g definiujemy (jak pamiętamy z logiki), jako funkcję $x \mapsto f(g(x))$. Zdefiniuj dwuargumentową procedurę `compose`, której wynikiem jest złożenie (jednoargumentowych) procedur przekazanych jej jako argumenty. Prześledź ewaluację wyrażeń `((compose square inc) 5)` oraz `((compose inc square) 5)` używając modelu podstawieniowego.

Ćwiczenie 3.

Zdefiniuj procedurę `(repeated p n)` obliczającą `n`-krotne złożenie procedury `p` z samą sobą. Nie używaj pomocniczych definicji procedur innych niż `compose` i

`identity`.

Ćwiczenie 4.

Zdefiniuj procedurę `product` analogiczną do procedury `sum` przedstawionej na wykładzie na dwa sposoby: jako procedurę generującą proces rekurencyjny i iteracyjny.

Użyj jednej z tych definicji do wyliczenia przybliżonej wartości π, używając wzoru $\frac{\pi}{4} = \frac{2 \cdot 4 \cdot 4 \cdot 6 \cdot 6 \cdot 8 \cdots}{3 \cdot 3 \cdot 5 \cdot 5 \cdot 7 \cdot 7 \cdots}$.

Ćwiczenie 5.

Zauważ że procedury `sum` i `product` są szczególnymi przypadkami jeszcze bardziej ogólnej procedury `accumulate`, wywoływanej w następujący sposób:

```
(accumulate combiner null-value term a next b)
```

W powyższym wyrażeniu `combiner` jest procedurą binarną określającą jak kolejny element ma być dołączony do dotychczas obliczonej wartości, `null-value` określa od jakiej wartości należy zacząć proces akumulacji, a pozostałe argumenty mają taką rolę jak w definicjach `sum` czy `product`. Zapisz definicję procedury `accumulate` na dwa sposoby, generujące odpowiednio proces rekurencyjny i iteracyjny, i pokaż jak zdefiniować `sum` i `product` jako szczególne przypadki akumulacji. Jakie własności muszą spełniać `combine` i `null-value` żeby wynik akumulacji z ich użyciem nie zależał od wyboru definicji (tj. był taki sam dla procesu iteracyjnego i rekurencyjnego).

Ułamki łańcuchowe

Interesującym pojęciem pojawiającym się w teorii liczb są *ułamki łańcuchowe* (ang. *infinite continued fractions*). W tej części listy zajmiemy się obliczaniem przybliżeń takich ułamków.

Nieskończonym ułamkiem łańcuchowym nazywamy wyrażenie postaci:

$$f = \cfrac{N_1}{D_1 + \cfrac{N_2}{D_2 + \cfrac{N_3}{D_3 + \ldots}}}$$

Jednym ze sposobów przybliżenia wartości ułamka łańcuchowego jest obcięcie jego rozwinięcia na określonej głębokości. Takie skończone rozwinięcie o

głębokości k ma wówczas postać:

$$f_k = \cfrac{N_1}{D_1 + \cfrac{N_2}{\ddots + \cfrac{N_k}{D_k + 0}}}$$

(co oczywiście daje nam $f_0 = 0$). Przykładowo jeśli wszystkie wyrazy ciągów N_i i D_i są równe 1, można łatwo pokazać że

$$\frac{1}{1 + \frac{1}{1+\ldots}} = \frac{1}{\varphi} \approx 0.618,$$

gdzie $\varphi = \frac{1+\sqrt{5}}{2}$ jest złotym podziałem. Kilka pierwszych wyrazów ciągu skończonych rozwinięć tego ułamka to

$$0, 1, \frac{1}{2}, \frac{2}{3}, \frac{3}{5}, \frac{5}{8}, \ldots$$

Ćwiczenie 6.

Załóżmy że procedury jednoargumentowe `num` i `den` określają odpowiednio kolejne wyrazy ciągów liczników i mianowników ułamka łańcuchowego, N_i i D_i. Zdefiniuj na dwa sposoby procedurę `cont-frac`, taką że `(cont-frac num den k)` obliczy `k`-ty wyraz ciągu skończonych rozwinięć ułamka łańcuchowego reprezentowanego przez `num` i `den`. Jeden z procesów generowanych przez Twoje definicje powinien być rekurencyjny, zaś drugi — iteracyjny. Przetestuj swoje procedury aproksymując wartość $\frac{1}{\varphi}$ obliczając wartość wyrażenia

```
(cont-frac (lambda (i) 1.0) (lambda (i) 1.0) k)
```

Ćwiczenie 7.

Liczbę π możemy zapisać używając ułamków łańcuchowych w następującej postaci:

$$\pi = 3 + \cfrac{1^2}{6 + \cfrac{3^2}{6 + \cfrac{5^2}{6+\ldots}}}.$$

Użyj tego rozwinięcia i procedury z poprzedniego zadania aby obliczyć przybliżoną wartość π. Którego przybliżenia potrzeba aby wartość była dokładna do czterech miejsc po przecinku?

Ćwiczenie 8.

Ułamki łańcuchowe często stosuje się do aproksymacji funkcji. Przykładowo, w 1770 niemiecki matematyk J.H. Lambert opublikował poniższą reprezentację funkcji arcus tangens:

$$\operatorname{arctg} x = \cfrac{x}{1 + \cfrac{(1x)^2}{3 + \cfrac{(2x)^2}{5 + \ldots}}}$$

Używając wcześniej zdefiniowanej procedury `cont-frac` zdefiniuj procedurę `atan-cf`, taką że `(atan-cf x k)` przybliża funkcję arcus tangens przez `k`-ty wyraz ciągu skończonych rozwinięć ułamka z powyższej reprezentacji. Przetestuj swoją procedurę dla różnych wartości `x`: użyj wbudowanej procedury `atan` aby sprawdzić jak dokładne są otrzymane przybliżenia.

Ćwiczenie 9.

Zajmiemy się teraz szczególnym przypadkiem ułamków łańcuchowych dla których ciągi N_i i D_i są ciągami stałymi. Wygodnie wtedy myśleć że skończone rozwinięcie takiego ułamka jest zbudowane z pewnej wartości bazowej za pomocą powtarzalnej operacji: mając podstawę B i wartości N i D, możemy zbudować wartość $\frac{N}{D+B}$. Łatwo zdefiniować procedurę `build` obliczającą taką wartość:

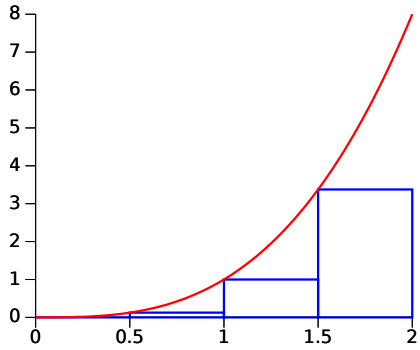
```
(define (build n d b)
  (/ n (+ d b)))
```

Aby obliczyć skończone rozwinięcie ułamka o głębokości dwa wystarczy wtedy obliczyć wartość `(build n d (build n d b))`.

Zdefiniuj czteroargumentową procedurę `repeated-build`, która stosując procedurę `repeated` z ćw. 3 obliczy k-ty wyraz ciągu skończonych rozwinięć ułamka łańcuchowego, k-krotnie stosując procedurę `build`. W szczególności, wyrażenie `(repeated-build 2 n d b)` powinno mieć tę samą wartość co wcześniejsze `(build n d (build n d b))`.

Zadanie domowe (na pracownię) nr 2

Metoda prostokątów to chyba najprostsza metoda przybliżania całki oznaczonej na funkcjach rzeczywistych: dzielimy przedział całkowania na p prostokątów, każdy „dotykający” wykresu funkcji „lewym” bokiem:



Źródło ilustracji: Wikipedia

Przybliżeniem całki jest wówczas suma pól tych prostokątów.

W tym zadaniu należy zaimplementować metodę prostokątów. Rozwiązanie powinno zawierać dwuargumentową procedurę `integral` taką, że w `(integral f prec)`:

- `f` to funkcja (czyli racketowa procedura), którą całkujemy
- `prec` to liczba prostokątów

Wynikiem powyższej aplikacji powinna być dwuargumentowa procedura, której dwa argumenty to początek i koniec przedziału całkowania. Np.

```
> (define foo (integral (lambda (x) 10) 1000))
> (foo 0 10)
99.9999999999986
> (define foo (integral (lambda (x) x) 1000))
> (foo 9 10)
9.499500000000001
> (foo 0 10)
49.95000000000001
> (define foo (integral sin 1000))
> (foo 0 (* pi 2))
1.0951119568461398e-16
> (foo 0 (/ pi 2))
0.9992143962198352
> ((integral tan 1000) (/ pi -2) (/ pi 2))
-51306101576015.1
```

W swojej implementacji użyj procedury `sum` zdefiniowanej na wykładzie (i w podręczniku).

W pliku zamieść też kilka testów. Opisz te z nich, w których metoda prostokątów daje niesatysfakcjonujące przybliżenie.

Uwaga! Plik o nazwie `solution.rkt` zawierający definicję procedury `integral` i przykłady testowe należy przesłać w systemie Web-CAT dostępnym na SKOS-owej stronie przedmiotu w *nieprzekraczalnym* terminie **15 marca 2021 r., godz. 05.30**. W pliku zamieść dodatkową klauzulę

```
(provide integral)
```

Sprawia ona, że procedura jest widoczna dla sprawdzaczki (dostępny też jest szablon rozwiązania na SKOS-ie). Pamiętaj o zasadach współpracy opisanych w regulaminie