Języki i paradygmaty programowania
Lista 1 (elementy języka Oz)

Przemysław Kobylański

Zaprogramuj w języku Oz rozwiązania poniższych zadań. Na ocenę dostateczną trzeba rozwiązać wszystkie zadania bez gwiazdek. Na ocenę dobrą trzeba dodatkowo rozwiązać wszystkie zadania z jedną gwiazdką. Na ocenę bardzo dobrą trzeba rozwiązać wszystkie zadania.

Część I
Programowanie deklaratywne

Zadanie 1.1

Zdefiniuj funkcję `{Reverse L}`, która dostarcza listę złożoną z elementów listy L ale w odwróconej kolejności.

Jaką czasową złożoność obliczeniową ma Twoja funkcja? Jeśli wyższą niż liniową względem długości listy L, to napisz rozwiązanie od nowa.

Zadanie 1.2

Zdefiniuj funkcję `{Merge L1 L2 L3}`, która scala trzy uporządkowane listy złożone z liczb w jedną uporządkowaną listę.

Zadanie 1.3

Zdefiniuj poniższe funkcje wykorzystując odpowiednio funkcję `{FoldL L F U}` albo `{FoldR L F U }`:

```
fun {Append L1 L2}
      case L1
      of nil then L2
      [] H1|T1 then H1|{Append T1 L2}
      end
end
fun {Reverse L}
      case L
      of nil then nil
      [] H|T then {Append {Reverse T} [H]}
      end
end
```

Zadanie 1.4*

Do zapisu binarnego drzewa uporządkowanego według kluczy zastosuj rekordy postaci:

```
tree(key:Key value:Value left:Left right:Right)
```

natomiast liście reprezentuj wartością `leaf`.

Zdefiniuj funkcję `{FindAll V T}`, która tworzy listę wszystkich kluczy z którymi wartość V występuje w drzewie T, przy czym klucze powinny być uporządkowane na tej liście.

Jaką czasową złożoność obliczeniową ma Twoja funkcja? Jeśli wyższą niż liniową względem liczby węzłów w drzewie T, to napisz rozwiązanie od nowa.

Zadanie 1.5**

Zdefiniuj funkcję `{Permutacje L}`, której wartością jest lista wszystkich permutacji elementów listy L.

Przykład obliczeń:

```
{Show {Permutacje [1 2 3]}}
[[1 2 3] [2 1 3] [3 2 1] [1 3 2] [3 1 2] [2 3 1]]
```

Uwaga: kolejność permutacji na wynikowej liście może być inna.

Część II
Współbieżność deklaratywna

Zadanie 2.1

Napisz procedurę `{Game Id1 Id2}`, która tworzy dwa wątki rozmawiające ze sobą za pomocą strumieni zrealizowanych listami otwartymi. Każdy z wątków realizuje grę jednego gracza (wątek pierwszy gracza o identyfikatorze Id1 a wątek drugi gracza o identyfikatorze Id2).

Każdy z graczy czeka aż wartość pojawi się mu na wejściu. Jeśli jest to `ping`, to wysyła na wyjście wartość `pong` a jeśli `pong`, to wysyła `ping`. Gdy gracz rozpozna co pojawiło się na jego wejściu, to drukuje swój identyfikator i otrzymaną wartość.

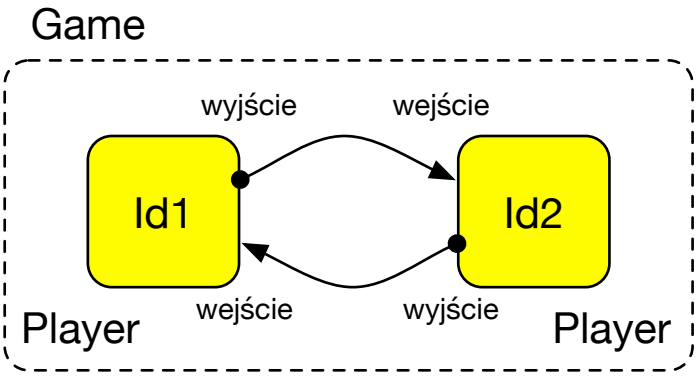
Na rysunku 1 przedstawiono schematycznie ideę programu.

Nie zapomnij "rzucić" pierwszemu graczowi piłeczkę (bez tego gra się nie rozpocznie).

Przykładowe uruchomienie:

```
{Game p1 p2}
p1#ping
p2#pong
p1#ping
p2#pong
p1#ping
p2#pong
p1#ping
p2#pong
...
```

Zwróć uwagę, że program działa w nieskończoność.



Rysunek 1: Dwaj gracze odbijający do siebie piłeczkę.

Zadanie 2.2*

Zdefiniuj procedurę `{NSort IN OUT}`, której parametrami są dwa strumienie liczb (wejściowy i wyjściowy).

Strumienie reprezentuj w postaci list otwartych (aby zakończyć strumień wystarczy otwarty ogon zunifikować z `nil`).

Działanie procedury `NSort` przypomina działanie tzw. *zero-time sorting network*[1].

Sieć taka składa się z połączonych ze sobą kolejnych węzłów, które można scharakteryzować następująco:

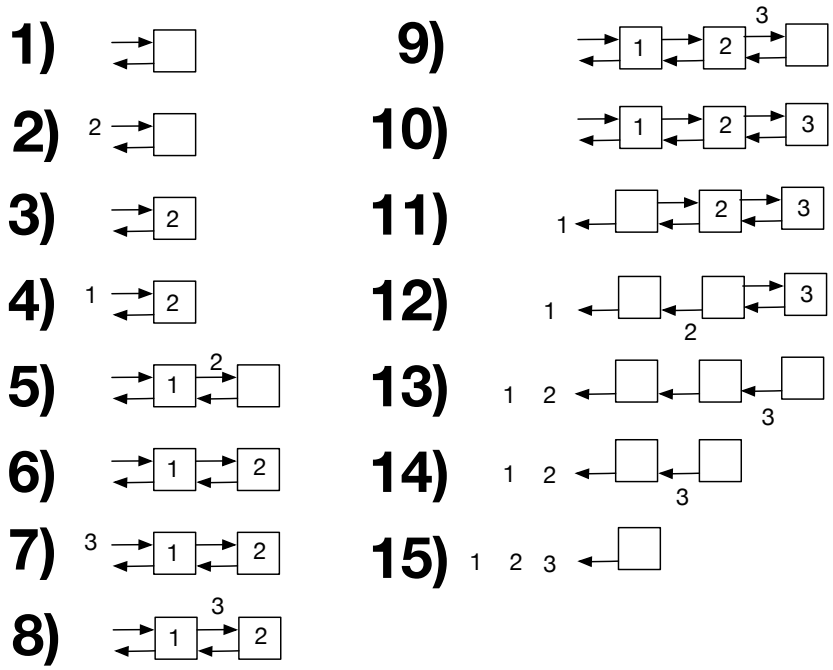
- Każdy węzeł ma własny akumulator, który zostaje zainicjowany pierwszą wartością jaka pojawiła się w strumieniu wejściowym do tego węzła.
- Kolejne liczby pojawiające się w strumieniu wejściowym są porównywane z zawartością akumulatora. Jeśli liczba jest większa lub równa, to zostaje ona wysłana do następnego węzła. Jeśli jest mniejsza od akumulatora, to zastępuje starą zawartość akumulatora po tym jak zostanie ona wysłany do następnego węzła.
- Jeśli zakończy się strumień wejściowy, to zawartość akumulatora wysyłana jest do strumienia wyjściowego, następnie kończy się strumień wychodzący do następnego węzła i zaczyna kopiować przychodzące z niego liczby do strumienia wyjściowego w ślad za wysłaną tam wcześniej zawartością akumulatora.
- Węzeł kończy pracę gdy zakończy się kopiowanie liczb z kolejnego węzła.

Na rysunku 2 przedstawiono działanie sieci `NSort`, która wejściowy ciąg liczb 2 1 3 przekształca w wyjściowy ciąg posortowany 1 2 3.

Zadanie 2.3** (zadanie 17 z rozdziału 4 książki [1])

Problem Hamminga dotyczy pierwszych n liczb całkowitych postaci $p_1^{a_1} p_2^{a_2} \cdots p_k^{a_k}$ dla $a_1, a_2, \ldots, a_k \geq 0$ przy użyciu pierwszych k liczb pierwszych $p_1, p_2, \ldots, p_k$. Napisz program, który rozwiązuje ten problem dla dowolnego n i k).

[1] W serwisie `http://books.google.pl` znajdziesz odpowiedni fragment książki [2] i możesz tam poczytać o podobnych sieciach.



Rysunek 2: Działanie sieci sortującej `NSort`.

Część III
Współbieżność z przesyłaniem komunikatów

Zadanie 3.1

Zamodelujmy rozmawiające dwie osoby A i B w następujący sposób używając dwóch funkcji f i g:

$$\begin{array}{ll} A: & f(x_0) \\ B: & g(f(x_0)) \\ A: & f(g(f(x_0))) \\ B: & g(f(g(f(x_0)))) \\ & \vdots \end{array}$$

Przy czym x_0, to inicjująca rozmowę wartość.

Zdefiniuj w języku Oz funkcję `{Rozmowca Fun Interlokutor}`, która tworzy punkt wejścia do portu rozmówcy nakładającego na bieżącą wartość swoją funkcję `Fun` i wysyłający tak wyliczoną nową wartość do interlokutora.

Rozmówca przed wysłaniem nowej wartości powinien wydrukować starą otrzymaną wartość za pomocą funkcji `Browse`.

Przykładowy dialog

Utworzenie dwóch rozmówców:

```
declare R1 R2 in
```

```
R1={Rozmowca fun {$ X} X+1 end R2}
R2={Rozmowca fun {$ X} X-1 end R1}
```

Zainicjowanie rozmowy:

```
{Send R1 0}
```

W oknie Browsera powinien pojawić się następujący nieskończony ciąg wartości:

```
0
1
0
1
0
1
0
1
...
```

Zadanie 3.2**

Zapoznaj się z przedstawionym w rozdziale 5 książki [1] systemem sterowania windami. Zaimplementuj go.

Literatura

[1] P. Van Roy, S. Haridi. Programowanie. Koncepcje, techniki i modele. Helion, 2005.

[2] R. Stephens. Essential Algorithms: A Practical Approach to Computer Algorithms. Wiley, 2013.