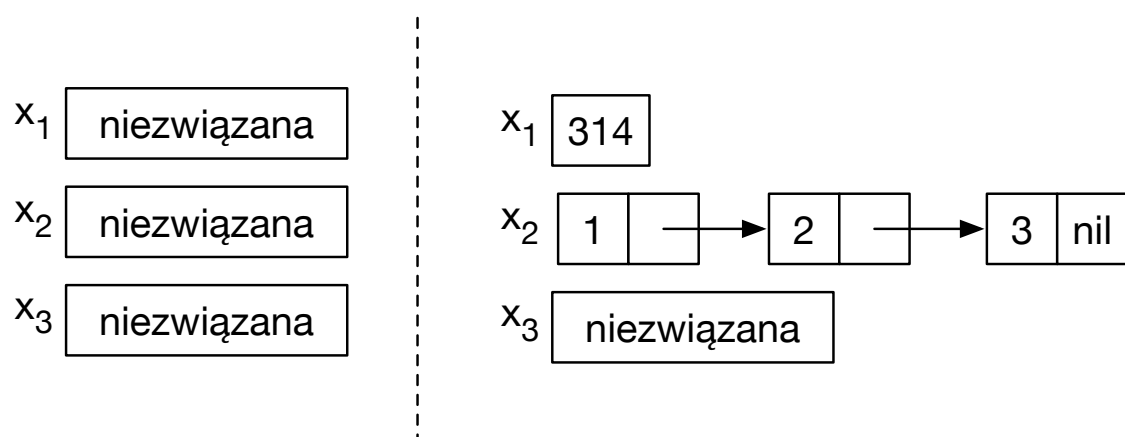


Przegląd paradygmatów na przykładzie języka **Oz**

- programowanie deklaratywne
- współbieżność deklaratywna
- współbieżność z przesyłaniem komunikatów

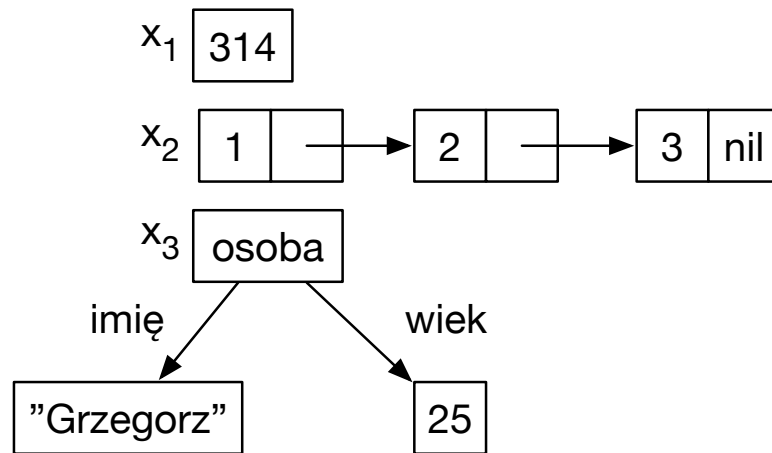
Programowanie deklaratywne

Obszar jednokrotnego przypisania (*single-assignment store*)



$\{x_1=314, x_2=[1\ 2\ 3], x_3\}$

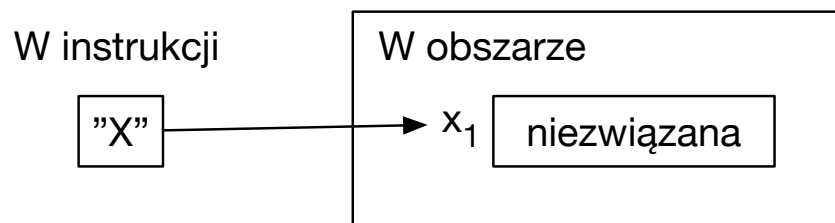
Programowanie deklaratywne



$\{x_1=314, x_2=[1\ 2\ 3], x_3=\text{osoba}(\text{imię}:\text{"Grzegorz"}\ \text{wiek}:24)\}$

Programowanie deklaratywne

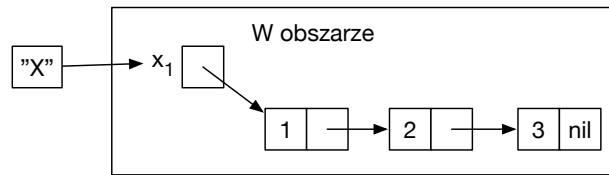
Identyfikator zmiennej umożliwia odwołanie się do wartości z obszaru przypisania.



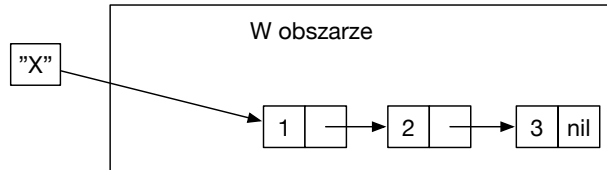
Odwzorowanie identyfikatorów zmiennych na elementy obszaru nazywa się **środowiskiem**.

$\{X \rightarrow x_1\}$

Programowanie deklaratywne



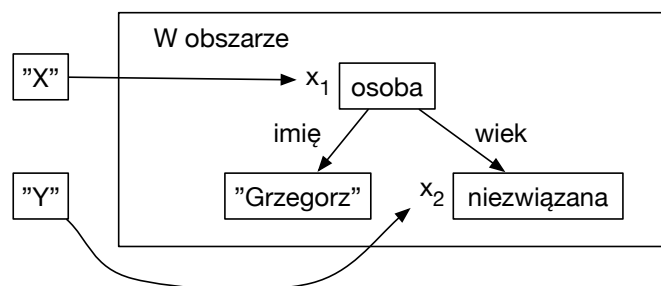
a) Identyfikator zmiennej odwołujący się do zmiennej związanej



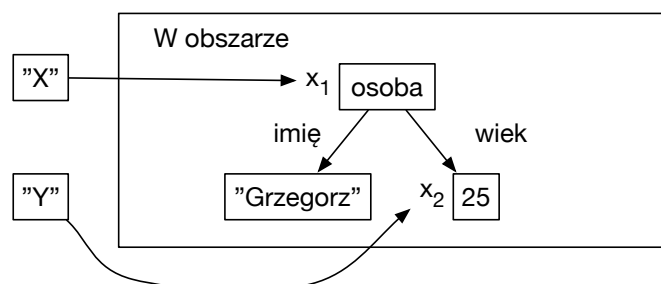
b) Identyfikator zmiennej odwołujący się wartości

Śledzenie łączy zmiennych związanych w celu otrzymania wartości nazywa się **dereferencją** (wyłuskaniem) i jest niewidoczne dla programisty.

Programowanie deklaratywne

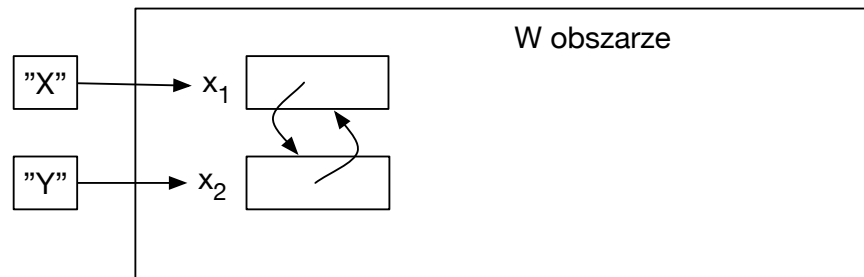


a) Wartość częściowa

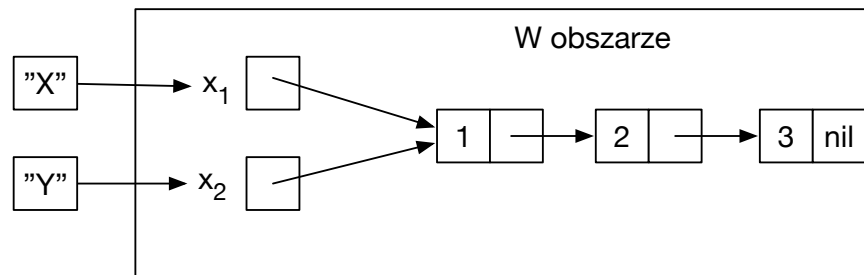


b) Wartość bez zmiennych niezwiązanych czyli wartość pełna

Programowanie deklaratywne



a) Dwie zmienne związane ze sobą



b) Obszar po związaniu jednej ze zmiennych

Programowanie deklaratywne

Po wykonaniu związania $X=Y$ otrzymujemy dwie zmienne związane.

Mówimy, że $\{x_1, x_2\}$ tworzy zbiór równoważności¹.

Kiedy jedna z równoważnych zmiennych zostaje związana, to wszystkie pozostałe zmienne widzą związaną.

¹ Formalnie: tworzą klasę równoważności ze względu na relację równoważności.

Programowanie deklaratywne

Zmienne przepływu danych

W programowaniu deklaratywnym tworzenie zmiennych i ich wiązanie wykonywane jest osobno. Co kiedy odwołamy się do zmiennej jeszcze nie związanej?

1. Wykonywanie jest kontynuowane bez komunikatu o błędzie.
Zawartość zmiennej jest niezdefiniowana, tzn. zawiera „śmieć”. (C++)
2. Wykonywanie jest kontynuowane bez komunikatu o błędzie.
Zawartość zmiennej jest inicjowana wartością domyślną. (Java w przypadku pól w obiektach albo tablicach)
3. Wykonywanie jest zatrzymane i pojawia się komunikat o błędzie.
(Prolog)
4. Wykonywanie niej jest możliwe ponieważ kompilator wykrył, że istnieje ścieżka wykonania wiodąca do użycia zmiennej bez jej zainicjowania.
(Java w przypadku zmiennych lokalnych)
5. Wykonywanie jest wstrzymane do momentu związania zmiennej a później kontynuowane. (Oz)

Programowanie deklaratywne

Język modelowy (*kernel language*)

```
<S> ::=  
  skip  
  | <S>1 <S>2  
  | local <X> in <S> end  
  | <X>1=<X>2  
  | <X>=<V>  
  | if <X> then <S>1 else <S>2 end  
  | case <X> of <pattern> then <S>1 else <S>2 end  
  | {<X> <y>1 ... <y>n}
```

Programowanie deklaratywne

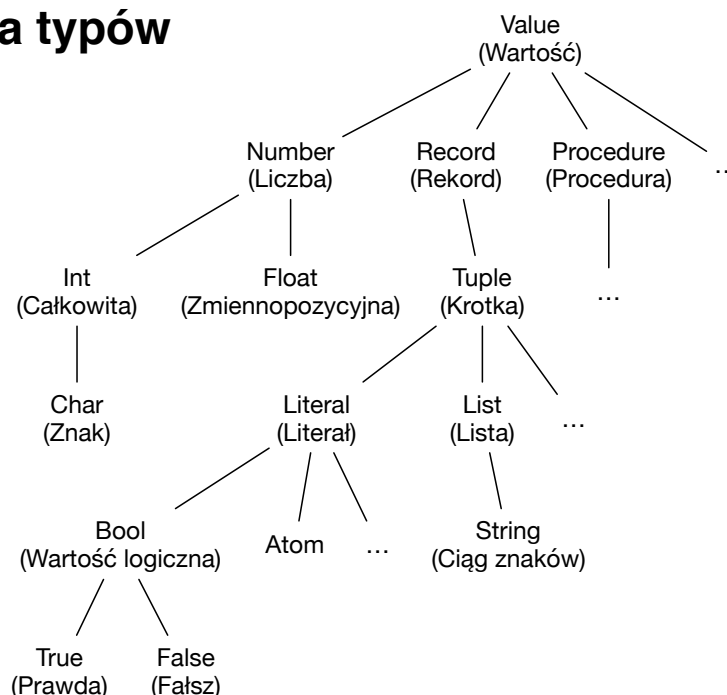
$\langle v \rangle ::= \langle \text{number} \rangle \mid \langle \text{record} \rangle \mid \langle \text{procedure} \rangle$
 $\langle \text{number} \rangle ::= \langle \text{int} \rangle \mid \langle \text{float} \rangle$
 $\langle \text{record} \rangle \langle \text{pattern} \rangle ::= \langle \text{literal} \rangle$
 $\mid \langle \text{literal} \rangle (\langle \text{feature} \rangle : \langle x \rangle_1 \dots \langle \text{feature} \rangle : \langle x \rangle_n)$
 $\langle \text{procedure} \rangle ::= \mathbf{proc} \{ \$ \langle x \rangle_1 \dots \langle x \rangle_n \} \langle s \rangle \mathbf{end}$
 $\langle \text{literal} \rangle ::= \langle \text{atom} \rangle \mid \langle \text{bool} \rangle$
 $\langle \text{feature} \rangle ::= \langle \text{atom} \rangle \mid \langle \text{bool} \rangle \mid \langle \text{int} \rangle$
 $\langle \text{bool} \rangle ::= \mathbf{true} \mid \mathbf{false}$

Składnia identyfikatora zmiennej:

- Wielka litera, po której występuje zero lub więcej znaków alfanumerycznych (litery, cyfry, znak podkreślenia)
- Dowolny ciąg znaków ujęty w odwrotne apostrofy.

Programowanie deklaratywne

Hierarchia typów



Programowanie deklaratywne

Typy podstawowe

- *Liczby*. Całkowite lub zmiennopozycyjne. Za minus należy używać znak tyldy (~).
- *Atomy*. Niepodzielne wartości będące rodzajem symbolicznych stałych. Ciąg znaków alfanumerycznych rozpoczynających się od małej litery albo dowolny ciąg znaków ujęty w apostrofy.
- *Wartości logiczne*. Symbol **true** albo **false**.
- *Rekordy*. Złożona struktura składająca się z etykiety, po której występuje zbiór par cech i identyfikatorów zmiennych. Cechami mogą być atomy, liczby całkowite lub wartości logiczne.
- *Krotki*. Krotka jest rekordem, którego cechami są liczby całkowite rozpoczynające się od 1 (w tym przypadku cechy nie muszą być podawane).

Programowanie deklaratywne

Typy podstawowe cd.

- *Listy*. Lista jest albo atomem **nil** albo krotką ' | ' (H T), gdzie T jest albo niezwiązane, albo związane z listą. Lukier składniowy:
 - Etykieta ' | ' może być zapisana jako operator wrostkowy, więc H|T oznacza to samo co ' | ' (H T).
 - Operator ' | ' wiąże prawostronnie, więc 1|2|3|nil oznacza to samo co 1|(2|(3|nil)).
 - Listy kończące się atomem **nil** mogą być zapisane przy użyciu nawiasów kwadratowych [...], więc zapis [1 2 3] oznacza to samo co 1|2|3|nil.
- *Ciągi znaków*. Ciąg znaków jest listą kodów znaków. Ciągi znaków zapisuje się za pomocą cudzysłowów, więc zapis "E=mc^2" oznacza to samo co [69 61 109 99 94 50].

Programowanie deklaratywne

Typy podstawowe cd.

- *Procedury*. Procedura jest wartością typu proceduralnego.

Instrukcja:

<x> = proc { \$ <y>₁ ... <y>_n } <s> end

wiąże <x> z nową wartością procedury. Oznacza to po prostu zadeklarowanie nowej procedury. Symbol \$ określa, że wartość procedury jest anonimowa, tj. tworzona bez związania jej z identyfikatorem. Możliwy jest zapis skrócony:

proc { <x> <y>₁ ... <y>_n } <s> end

Taki zapis skrócony jest czytelniejszy ale zaciemnia rozróżnienie między utworzeniem wartości a związaniem jej z identyfikatorem.

Programowanie deklaratywne

Przykłady operacji podstawowych

Operacja	Opis	Typ argumentu
A==B	Porównanie równości	Value
A\=B	Porównanie nierówności	Value
{IsProcedure P}	Sprawdzenie czy procedura	Value
A<=B	Porównanie mniejszy niż lub równy	Number lub Atom
A<B	Porównanie mniejszy niż	Number lub Atom
A>=B	Porównanie większy niż lub równy	Number lub Atom
A>B	Porównanie większy niż	Number lub Atom
A+B	Dodawanie	Number
A-B	Odejmowanie	Number
A*B	Mnożenie	Number

Programowanie deklaratywne

Przykłady operacji podstawowych cd.

Operacja	Opis	Typ argumentu
A div B	Dzielenie	Int
A mod B	Modulo	Int
A/B	Dzielenie	Float
{Arity R}	Krotność	Record
{Label R}	Etykieta	Record
R.F	Wybór pola	Record

Programowanie deklaratywne

Proste wykonanie

```
local A B C D in  
  A=11  
  B=2  
  C=A+B  
  D=C*C  
end
```

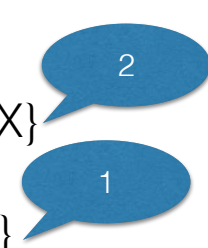


```
local A in  
  local B in  
    local C in  
      local D in  
        A=11  
        B=2  
        C=A+B  
        D=C*C  
      end  
    end  
  end  
end
```

Programowanie deklaratywne

Statyczne wyznaczanie zakresu

```
local X in  
  X=1  
  local X in  
    X=2  
    {Browse X}  
  end  
  {Browse X}  
end
```



The diagram shows two nested blue speech bubbles. The inner bubble, labeled '2', points to the inner 'local X in' block. The outer bubble, labeled '1', points to the outer 'local X in' block. This illustrates how static scope resolution determines the binding of the variable X based on its lexical position in the code.

Programowanie deklaratywne

Procedury

```
proc {Max X Y ?Z}  
  if X>=Y then Z=X else Z=Y end  
end
```

Procedury z odwołaniami zewnętrznymi

```
proc {LB X ?Z}  
  if X>=Y then Z=X else Z=Y end  
end
```


Programowanie deklaratywne

```
local Y LB in  
  Y=10  
  proc {LB X ?Z}  
    if X>=Y then Z=X else Z=Y end  
  end  
  local Y=15 Z in  
    {LB 5 Z}  
  end  
end
```

Programowanie deklaratywne

Dynamiczne a statyczne określanie zasięgu

```
local P Q in  
  proc {Q X} {Browse stat(X)} end  
  proc {P X} {Q X} end  
  local Q in  
    proc {Q X} {Browse dyn(X)} end  
    {P hello}  
  end  
end
```



stat(hello)

Oryginalna wersja języka Lisp obsługiwała **dynamiczne** wyznaczanie zakresu. Języki Common Lisp i Scheme domyślnie obsługują **statyczne** wyznaczanie zakresu.

Programowanie deklaratywne

Zachowanie w przypadku przepływu danych

```
local X Y Z in  
  X=10  
  if X>=Y then Z=X else Z=Y end  
end
```

Nie jest możliwe określenie wartości porównania $X \geq Y$ ale nie jest zgłaszany błąd tylko wykonanie instrukcji warunkowej zostaje wstrzymane do chwili gdy Y zostanie związane.

Programowanie deklaratywne

Przekład na język modelowy

```
local Max C in  
  proc {Max X Y ?Z}  
    if X>=Y then Z=X else Z=Y end  
  end  
  {Max 3 5 C}  
end
```

Programowanie deklaratywne

Przekład na język modelowy cd.

```
local Max in
  local A in
    local B in
      local C in
        Max = proc {$ X Y Z}
          local T in
            T=(X>=Y)
            if T then Z=X else Z=Y end
          end
        end
      A=3
      B=5
      {Max A B C}
    end
  end
end
end
```

Programowanie deklaratywne

Od języka modelowego do języka praktycznego

Programy w języku modelowym są zbyt rozwlekłe. Można tę wadę wyeliminować dodając **lukier syntaktyczny** i **abstrakcje lingwistyczne**.

Udogodnienia składniowe

Zagnieżdżone wartości częściowe:
zamiast

```
local A B in A="Grzegorz" B=25 X=osoba(imię:A wiek:B) end
```

wygodniej

```
X=osoba(imię:"Grzegorz" wiek:25)
```

Programowanie deklaratywne

Niejawna inicjalizacja zmiennych:

zamiast

local X **in** X=<expression> <statement> **end**

wygodniej

local X=<expression> **in** <statement> **end**

Przypadek ogólny ma postać

local <pattern>=<expression> **in** <statement> **end**

Przykład:

local tree(key:A left:B right:C value:D)=T **in** <statement> **end**

Programowanie deklaratywne

Zagnieżdżone instrukcje **if**:

<statement> ::= **if** <expression> **then** <inStatement>
 { **elseif** <expression> **then** <inStatement> }
 [**else** <inStatement>] **end**

<inStatement> ::= [{ <declarationPart> }+ **in**] <statement>

Programowanie deklaratywne

Zagnieżdżone instrukcje **case**:

```
<statement> ::= case <expression>
               of <pattern> [ andthen <expression> ]
               then <inStatement>
               { '[' <pattern> [ andthen <expression> ]
                 then <inStatement> }
               [ else <inStatement> ] end
<pattern> ::= <variable> | <atom> | <int> | <float>
             | <string> | unit | true | false
             | <label> '(' { [ <feature> ':' ] <pattern> } [ '...' ] ')'
             | <pattern> <consBinOp> <pattern>
             | '[' { <pattern> }+ ']'
<consBinOp> ::= '#' | '|'
```

Programowanie deklaratywne

Przykład:

case Xs#Ys

of nil#Ys **then** <S>₁

 [] Xs#nil **then** <S>₂

 [] (X|Xr)#(Y|Yr) **andthen** X<=Y **then** <S>₃

else <S>₄

end

case Xs **of** nil **then** <S>₁

else

case Ys **of** nil **then** <S>₂

else

case Xs **of** X|Xr **then**

case Ys **of** Y|Yr **then**

if X<=Y **then** <S>₃ **else** <S>₄ **end**

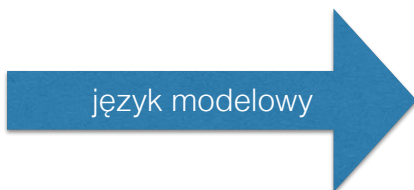
else <S>₄ **end**

else <S>₄ **end**

end

end

język modelowy



Programowanie deklaratywne

Funkcje

fun {F X1 ... XN} <statement> <expression> **end**



proc {F X1 ... XN ?R} <statement> R=<expression> **end**

Programowanie deklaratywne

Przykład:

fun {Max X Y}
 if X>=Y **then** X **else** Y **end**
end



proc {Max X Y ?R}
 R = **if** X>=Y **then** X **else** Y **end**
end

jeśli treść funkcji jest
instrukcją **if**, każda
alternatywa tej instrukcji może
kończyć się wyrażeniem

Programowanie deklaratywne

Wywołanie funkcji

Wywołanie funkcji $\{F X1 \dots XN\}$ jest tłumaczone na wywołanie procedury $\{F X1 \dots XN R\}$.

Przykład:

Wywołanie $\{Q \{F X1 \dots XN\} \dots\}$ jest tłumaczone na:

local R **in**

$\{F X1 \dots XN R\}$

$\{Q R \dots\}$

end

Programowanie deklaratywne

Wywołanie funkcji w strukturach danych

Można wywołać funkcję w ramach struktury danych (rekordu, krotki lub listy).

Przykład:

$Ys = \{F X\} | \{Map Xr F\}$

jest tłumaczone na:

local Y Yr **in**

$Ys = Y | Yr$

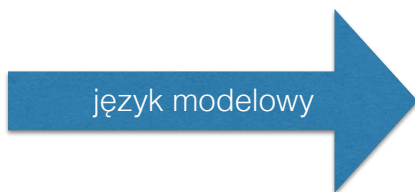
$\{F X Y\}$

$\{Map Xr F Yr\}$

end

Programowanie deklaratywne

```
fun {Map Xs F}  
  case Xs  
  of nil then nil  
  [] X|Xr then {F X}|{Map Xr F}  
end  
end
```



```
proc {Map Xs F ?Ys}  
  case Xs of nil then Ys=nil  
  else case Xs of X|Xr then  
    local Y Yr in  
      Ys=Y|Yr {F X Y} {Map Xr F Yr}  
    end  
  end end  
end
```

Programowanie deklaratywne

Interfejs interaktywny (zmienne globalne)

```
<interStatement> ::=  
  <statement>  
  | declare { <declarationPart> }+ [ <interStatement> ]  
  | declare { <declarationPart> }+ in <interStatement>  
<declarationPart> ::=  
  <variable> | <pattern> '=' <expression> | <statement>
```

Przykład:

```
declare X Y
```

```
X=25
```

```
declare A
```

```
A=osoba(wiek:X)
```

```
declare X Y
```

nowa zmienna X ale
wartość 25 nadal jest
osiągalna przez
zmienną globalną A

Programowanie deklaratywne

Wyjątki

```
<S> ::= ...  
      | try <S>1 catch <x> then <S>2 end  
      | raise <x> end
```

- Instrukcja w której wywołano wyjątek zostaje anulowana.
- Wyjątkiem może być każda wartość częściowa.
- Gdy wyjątek jest zmienną niezwiązaną, to jego zgłoszenie może być współbieżne z jego określeniem (może być nawet przechwycony zanim będzie wiadomo, o który wyjątek chodzi!).
- Jest to rozsądne tylko w przypadku języków ze zmiennymi przepływu danych.

Programowanie deklaratywne

Przykład:

```
fun {Eval E}  
  if {IsNumber E} then E  
  else  
    case E  
    of plus(X Y) then {Eval X}+{Eval Y}  
    [] times(X Y) then {Eval X}*{Eval Y}  
    else raise illFormedExpr(E) end  
  end  
end
```

Programowanie deklaratywne

Przykład cd.

```
try
  {Browse {Eval plus(plus(5 5) 10)}}
  {Browse {Eval times(6 11)}}
  {Browse {Eval minus(7 10)}}
catch illFormedExpr(E) then
  {Browse '*** Błędne wyrażenie '#E#' ***'}
end
```

Programowanie deklaratywne

Instrukcja **try** może określać klauzulę **finally**, która jest zawsze wykonywana bez względu na to, czy instrukcja zgłosi wyjątek czy nie.

Przykład:

```
try
  {ProcessFile F}
finally {CloseFile F} end
```

Programowanie deklaratywne

Algorytm unifikacji, struktury cykliczne i równość

Operacje związania:

- $\text{bind}(\text{ES}, \langle v \rangle)$ wiąże wszystkie zmienne ze zbiorów równoważnych zmiennych ES z wartością $\langle v \rangle$.
- $\text{bind}(\text{ES}_1, \text{ES}_2)$ łączy dwa zbiory równoważnych zmiennych ES_1 i ES_2 .

Programowanie deklaratywne

Definicja operacji $\text{unify}(x, y)$ bez struktur cyklicznych:

1. Jeśli x jest w zbiorze ES_x a y w zbiorze ES_y , to wykonaj $\text{bind}(\text{ES}_x, \text{ES}_y)$.
2. Jeśli x jest w zbiorze ES_x a y jest określone, to wykonaj $\text{bind}(\text{ES}_x, y)$.
3. Jeśli y jest w zbiorze ES_y a x jest określone, to wykonaj $\text{bind}(\text{ES}_y, x)$.
4. Jeśli x jest związane z $r(f_1:x_1 \dots f_n:x_n)$ a y jest związane z $r'(f'_1:y_1 \dots f'_m:y_m)$ i $r \neq r'$ lub $\{f_1, \dots, f_n\} \neq \{f'_1, \dots, f'_m\}$, to zgłoś wyjątek **failure**.
5. Jeśli x jest związane z $r(f_1:x_1 \dots f_n:x_n)$ a y jest związane z $r(f_1:y_1 \dots f_n:y_n)$, to dla i od 1 do n wykonaj $\text{unify}(x_i, y_i)$.

Programowanie deklaratywne

Uwzględnienie struktur cyklicznych.

- Niech **M** będzie pustą tabelą.
- Wykonaj **unify**"(x, y).

Operacja **unify**"(x, y) najpierw sprawdza czy para (x, y) występuje w tabeli **M**.


1. Jeśli występuje, to kończy pracę.
2. Jeśli nie występuje, to wywołuje operację **unify**(x, y), w której rekurencyjne wywołania **unify** zastąpiono wywołaniami **unify**".

Programowanie deklaratywne

Sprawdzenie równości i nierówności

$X=Y$ zachodzi gdy obie struktury są identyczne.

Możliwe jest wcześniejsze stwierdzenie, że zachodzi $X \neq Y$ nawet gdy obie struktury są jeszcze niepełne.

declare L1 L2 X in	declare L1 L2 X Y in	declare L1 L2 X in
L1=[1] L2=[X] {Browse L1==L2}	L1=[X] L2=[Y] {Browse L1==L2}	L1=[1 2] L2=[2 X] {Browse L1==L2}
nic się nie pojawia bo czeka na związanie X	X=Y po unifikacji pojawia się true	 od razu pojawia się false

Programowanie deklaratywne

Techniki programowania: obliczenia iteracyjne

$$S_0 \rightarrow S_1 \rightarrow \dots \rightarrow S_{\text{final}}$$

```
fun {Iterate Si}  
  if {IsDone Si} then Si  
  else Si+1 in  
    Si+1={Transform Si}  
    {Iterate Si+1}  
  end  
end
```



```
fun {Iterate S IsDone Transform}  
  if {IsDone S} then S  
  else S1 in  
    S1={Transform S}  
    {Iterate S1 IsDone Transform}  
  end  
end
```

Programowanie deklaratywne

Techniki programowania: obliczenia iteracyjne

```
fun {Sqrt X}  
  {Iterate  
    1.0  
    fun {$ G} {Abs X-G*G}/X<0.00001 end  
    fun {$ G} (G+X/G)/2.0 end  
  }  
end
```

Programowanie deklaratywne

Techniki programowania: konwersja rekurencji na iterację

```
fun {Fact N}  
  if N==0 then 1  
  else N*{Fact N-1}  
  end  
end
```



```
fun {Fact N}  
  fun {IterFact N Acc}  
    if N==0 then Acc  
    else {IterFact N-1 N*Acc}  
    end  
  end  
in  
  {IterFact N 1}  
end
```

Programowanie deklaratywne

Techniki programowania: operacje na listach

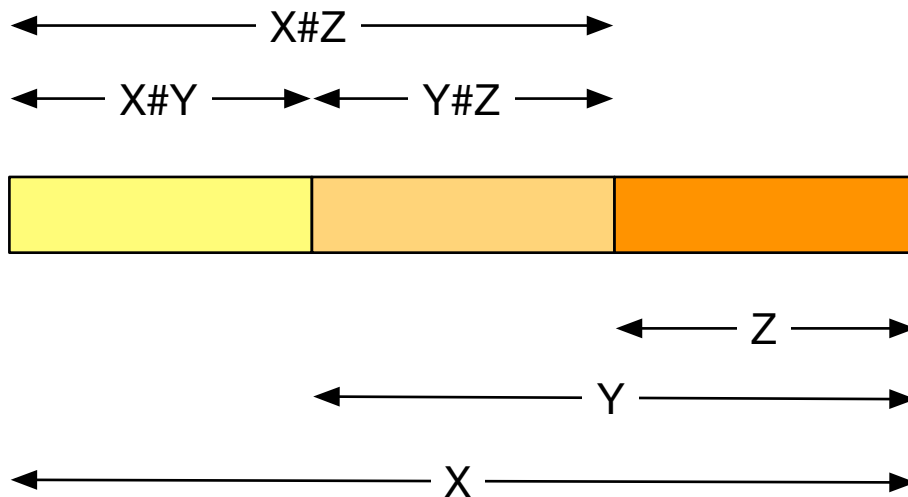
```
fun {Length Xs}  
  case Xs of nil then 0  
  [] _|Xr then 1+{Length Xr}  
  end  
end
```



```
fun {Length Xs}  
  fun {LengthIter Xs Acc}  
    case Xs of nil then Acc  
    [] _|Xr then {LengthIter Xr Acc+1}  
    end  
  end  
in  
  {LengthIter Xs 0}  
end
```


Programowanie deklaratywne

Techniki programowania: listy różnicowe



Programowanie deklaratywne

Techniki programowania: listy różnicowe

```
fun {AppendD D1 D2}  
  X#Y=D1  
  Y#Z=D2  
in  
  X#Z  
end
```

```
declare D1 D2 X Y in  
D1=(1|2|3|X)#X  
D2=(4|5|6|7|Y)#Y  
{Browse {AppendD D1 D2}}
```

(1|2|3|4|5|6|7|_)#_

Programowanie deklaratywne

Techniki programowania: uporządkowane drzewa binarne

```
fun {Insert K V T}  
  case T  
  of leaf then tree(K V leaf leaf)  
    [] tree(K1 V1 T1 T2) andthen K==K1 then  
      tree(K V T1 T2)  
    [] tree(K1 V1 T1 T2) andthen K<K1 then  
      tree(K1 V1 {Insert K V T1} T2)  
    [] tree(K1 V1 T1 T2) andthen K>K1 then  
      tree(K1 V1 T1 {Insert K V T2})  
  end  
end
```

Programowanie deklaratywne

Techniki programowania: uporządkowane drzewa binarne

```
fun {Lookup K T}  
  case T  
  of leaf then notfound  
    [] tree(K1 V T1 T2) andthen K==K1 then found(V)  
    [] tree(K1 V T1 T2) andthen K<K1 then {Lookup K T1}  
    [] tree(K1 V T1 T2) andthen K>K1 then {Lookup K T2}  
  end  
end
```

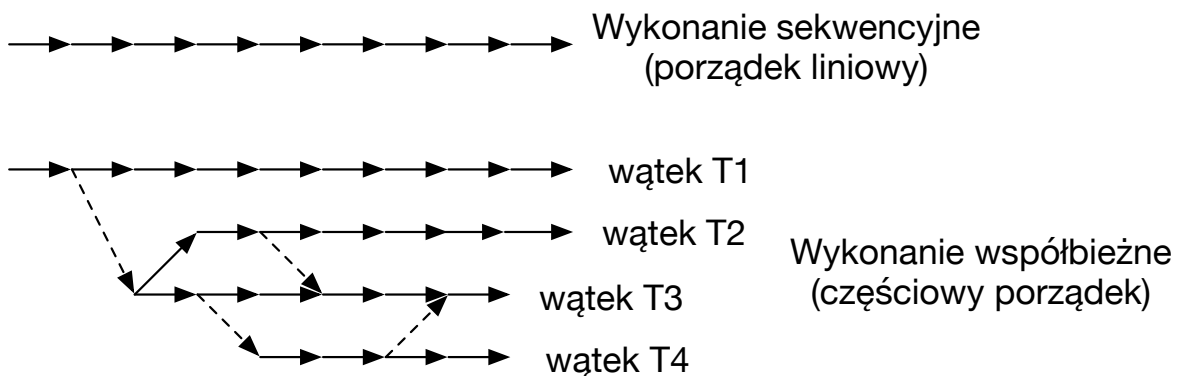
Współbieżność deklaratywna

Język modelowy

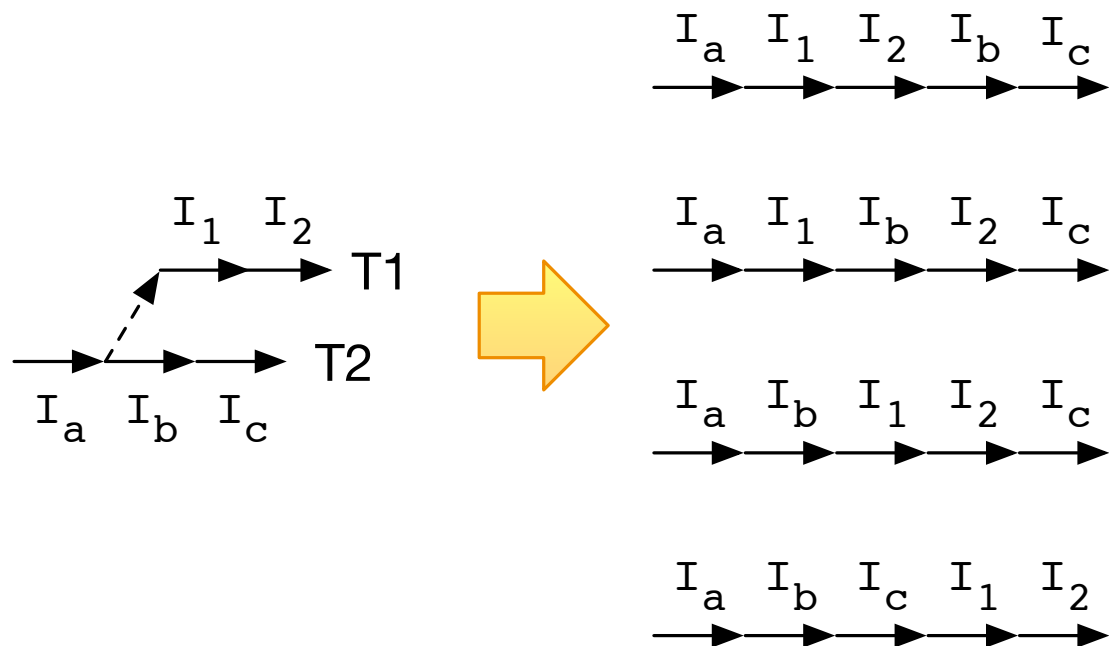
```
<S> ::=  
  skip  
  | <S>1 <S>2  
  | local <X> in <S> end  
  | <X>1 = <X>2  
  | <X> = <V>  
  | if <X> then <S>1 else <S>2 end  
  | case <x> of <pattern> then <S>1 else <S>2 end  
  | {<X> <y>1 ... <y>n}  
  | thread <S> end
```

Współbieżność deklaratywna

Porządek przyczynowy



Współbieżność deklaratywna



Współbieżność deklaratywna

Częściowe zakończenie

```
fun {Double Xs}  
  case Xs of X|Xr then 2*X|{Double Xr} end  
end
```

- Obliczenia powyższej funkcji nigdy się nie zakończą.
- Jeśli strumień wejściowy **Xs** przestanie rosnać, to obliczenie się zakończy.
- Obliczenie osiągnie **częściowe zakończenie**, ponieważ jeśli znowu strumień wejściowy zacznie znowu rosnać, to obliczenia zostaną wznowione aż do następnego częściowego zakończenia.

Współbieżność deklaratywna

Logiczna równoważność

$$X=1 \ Y=X \equiv Y=X \ X=1$$

w obu przypadkach
X i Y są związane z
wartością 1

$$X=\text{foo}(Y \ W) \ Y=Z \equiv X=\text{foo}(Z \ W) \ Y=Z$$

Zbiór wiązań w obszarze jednokrotnego przypisania nazywamy ograniczeniem. Niech $\text{values}(x, c)$ będzie zbiorem wartości dla zmiennej x w ograniczeniu c , które nie naruszają ograniczenia c .

Ograniczenia c_1 i c_2 są logicznie równoważne, gdy:

1. Zawierają te same zmienne.
2. Dla każdej zmiennej x , $\text{values}(x, c_1) = \text{values}(x, c_2)$.

Współbieżność deklaratywna

Deklaratywna współbieżność

Współbieżny program jest **deklaratywny** jeśli dla dowolnych wejść zachodzi poniższy warunek.

Wszystkie wykonania, dla danego zbioru wejść, mają jeden z dwóch rezultatów:

1. wszystkie nie kończą się lub
2. wszystkie kończą się osiągając częściowe zakończenie i dają logicznie równoważne rezultaty.

niezauważalny
niedeterminizm

Współbieżność deklaratywna

Awaria (*failure*)

Awaria jest nieprawidłowym zakończeniem deklaratywnego programu osiągnięte przez umieszczenie niezgodnych (konfliktowych) informacji w obszarze przypisania.

```
thread X=1 end  
thread Y=2 end  
thread X=Y end
```

każde wykonanie
osiąga
niepowodzenie

Współbieżność deklaratywna

Zamknięcie awarii (ukrywanie niedeterminizmu)

```
declare X Y  
local X1 Y1 S1 S2 S3 in  
  thread  
    try X1=1 S1=ok catch _ then S1=error end  
  end  
  thread  
    try Y1=2 S2=ok catch _ then S2=error end  
  end  
  thread  
    try X1=Y1 S3=ok catch _ then S3=error end  
  end  
  if S1==error or S2==error or S3==error then  
    X=1 Y=1 % wartości domyślne dla X i Y  
  else  
    X=X1 Y=Y1 end  
end
```

Współbieżność deklaratywna

Tworzenie nowego wątku

thread

proc {Count N} **if** N>0 **then** {Count N-1} **end end**

in

{Count 1000000}

end



declare X in

X = **thread** 10*10 **end** + 100*100

{Browse X}

declare X in

local Y in

thread Y = 10*10 **end**

X = Y + 100*100

end

{Browse X}

Współbieżność deklaratywna

Przykład: liczby Fibonacciego

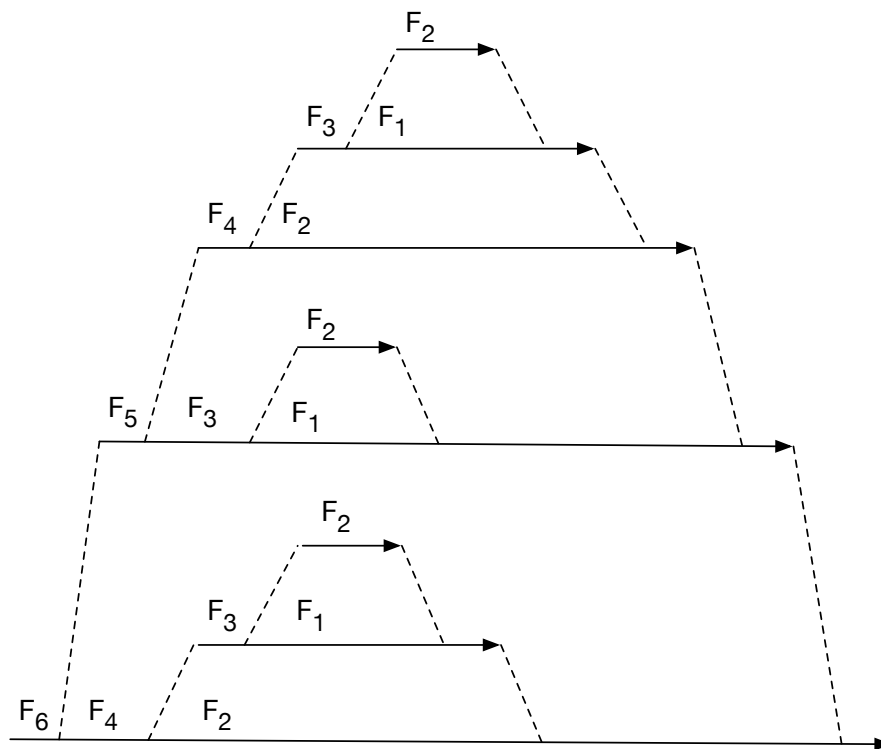
fun {Fib X}

if X=<2 **then** 1

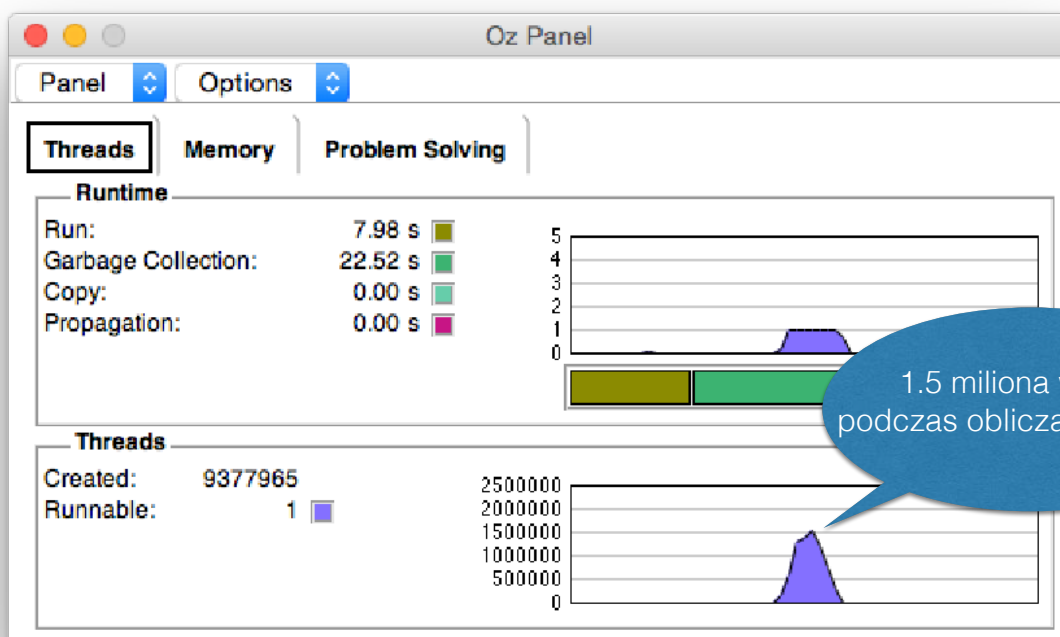
else thread {Fib X-1} **end** + {Fib X-2} **end**

end

Współbieżność deklaratywna



Współbieżność deklaratywna



Współbieżność deklaratywna

Strumienie w postaci list otwartych

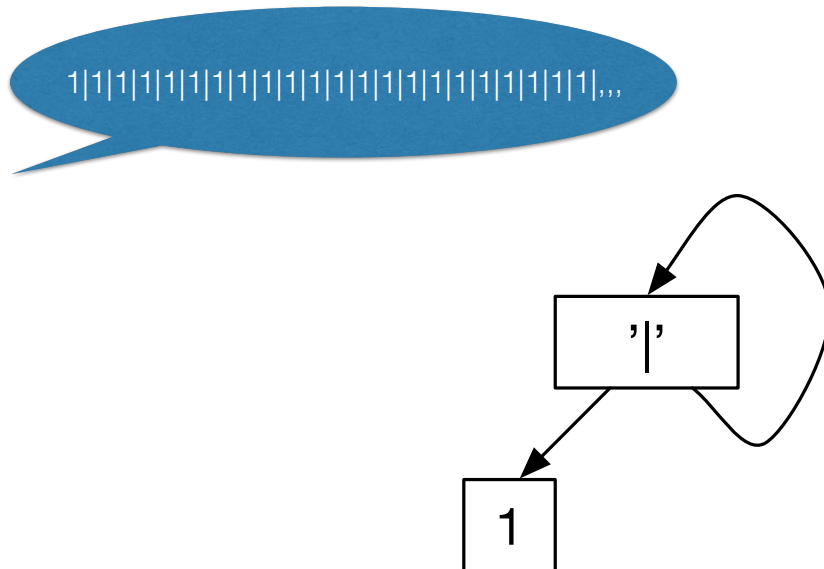
```
X=1|X1
X1=2|X2
X2=3|X3
X3=4|X4
X4=nil
```

Zmienna X zostaje związana z listą, na której początkowo jest tylko liczba 1. Później dochodzą jeszcze liczby 2, 3 i 4. Na koniec lista zostaje zamknięta (nie wa nieustalonego ogona).

Współbieżność deklaratywna

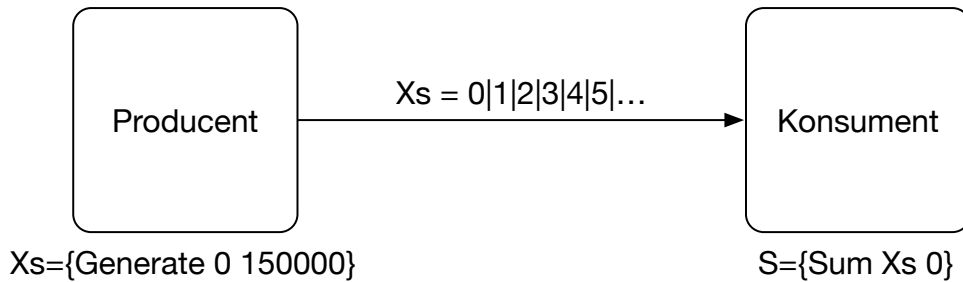
Nieskończony strumień liczb

X=1|X
{Browse X}



Współbieżność deklaratywna

Schemat producenta-konsumenta



Współbieżność deklaratywna

```
fun {Generate N Limit}
  if N<Limit then
    N|{Generate N+1 Limit}
  else nil
end
fun {Sum Xs A}
  case Xs
  of X|Xr then {Sum Xr A+X}
  [] nil then A
  end
end
local Xs S in
  thread Xs={Generate 0 150000} end % wątek producenta
  thread S={Sum Xs 0} end % wątek konsumenta
  {Browse S}
end
```

Współbieżność deklaratywna

Odczyt wielokrotny

```
local Xs S1 S2 S3 in  
  thread Xs={Generate 0 150000} end  
  thread S1={Sum Xs 0} end  
  thread S2={Sum Xs 0} end  
  thread S3={Sum Xs 0} end  
end
```

wątek każdego
konsumenta pobiera
elementy strumienia
niezależnie

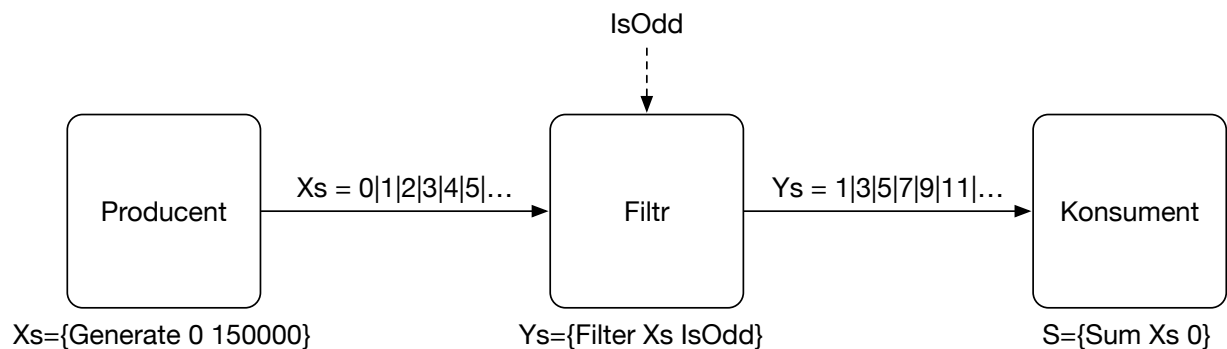
Współbieżność deklaratywna

Przetworniki i filtry

```
fun {Filter Xs F}  
  case Xs  
  of nil then nil  
  [] X|Xr andthen {F X} then X|{Filter Xr F}  
  [] X|Xr then {Filter Xr F}  
  end  
end
```

```
fun {IsOdd X} X mod 2 \= 0 end
```

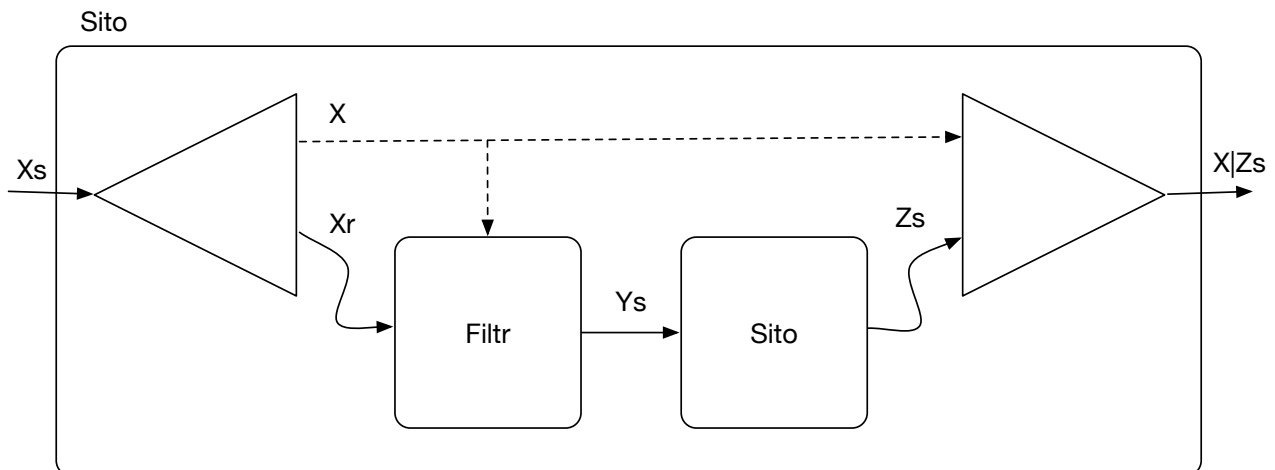
Współbieżność deklaratywna



```
local Xs Ys S in  
  thread Xs={Generate 0 150000} end  
  thread Ys={Filter Xs IsOdd} end  
  thread S={Sum Ys 0} end  
  {Browse S}  
end
```

Współbieżność deklaratywna

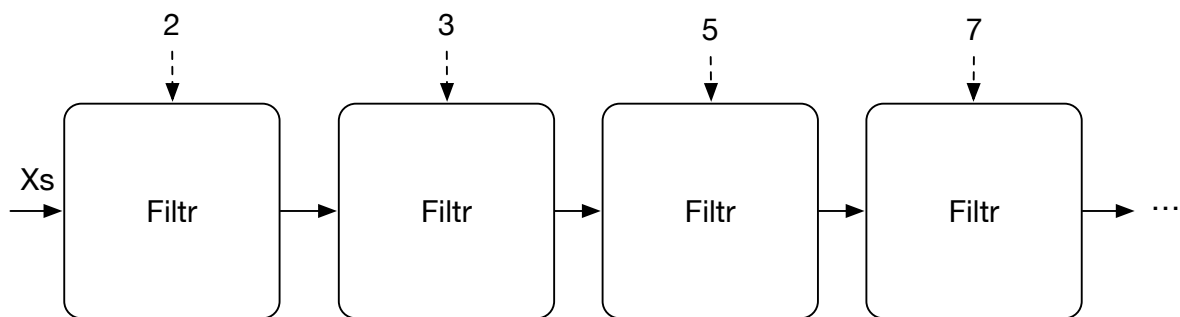
Sito Eratostenesa



Współbieżność deklaratywna

```
fun {Sieve Xs}  
  case Xs  
  of nil then nil  
  [] X|Xr then Ys in  
    thread Ys={Filter Xr fun {$ Y} Y mod X  $\neq$  0 end} end  
    X|{Sieve Ys}  
  end  
end
```

Współbieżność deklaratywna



Potok filtrów

Współbieżność deklaratywna

Wykonywanie leniwe

- ewaluacja gorliwa (*data-driven evaluation*)
- ewaluacja leniwa (*demand-driven evaluation*)

Współbieżność deklaratywna


Wykonywanie leniwe

```
fun lazy {F1 X} 1+X*(3+X*(3+X)) end  
fun lazy {F2 X} Y=X*X in Y*Y end  
fun lazy {F3 X} (X+1)*(X+1) end  
A={F1 10}  
B={F2 20}  
C={F3 30}  
D=A+B
```

Współbieżność deklaratywna

Wyzwalacze potrzeby

```
<S> ::=  
  skip  
  | <S>1 <S>2  
  | local <X> in <S> end  
  | <X>1=<X>2  
  | <X>=<V>  
  | if <x> then <S>1 else <S>2 end  
  | case <x> of <pattern> then <S>1 else <S>2 end  
  | {<x> <y>1 ... <y>n}  
  | thread <S> end  
  | {ByNeed <x> <y>}
```



Współbieżność deklaratywna

Instrukcja {ByNeed P Y} ma ten sam efekt, co instrukcja **thread** {P Y} **end** ale wywołanie {P X} zostanie wykonane tylko wtedy, gdy potrzebna jest wartość Y.

```
declare Y  
{ByNeed proc {$ A} A=111*111 end Y}  
{Browse Y}  
declare Z  
Z=Y+1
```

w oknie Browser
pojawia się
wartość 12321

Współbieżność deklaratywna

Problem Hamminga

Wygenerować pierwszych n liczb całkowitych postaci $2^a 3^b 5^c$, gdzie $a, b, c \geq 0$.

Idea: generować liczby całkowite w porządku rosnącym w potencjalnie nieskończonym strumieniu.

```
fun lazy {Times N H}
  case H of X|H2 then N*X|{Times N H2} end
end
```

dla nieskończonego strumienia
liczb H, wartością jest
nieskończony strumień liczb N
razy większych

Współbieżność deklaratywna

```
fun lazy {Merge Xs Ys}
  case Xs#Ys of (X|Xr)#(Y|Yr) then
    if X<Y then X|{Merge Xr Ys}
    elseif X>Y then Y|{Merge Xs Yr}
    else X|{Merge Xr Yr}
  end
end
end
```

leniwe scalenie dwóch
uporządkowanych i
nieskończonych strumieni liczb
w jeden nieskończony i
uporządkowany strumień liczb

Współbieżność deklaratywna

```
H=1|{Merge {Times 2 H}
      {Merge {Times 3 H}
             {Times 5 H}}}}
{Browse H}
```

wyświetli się 1|_<Future>

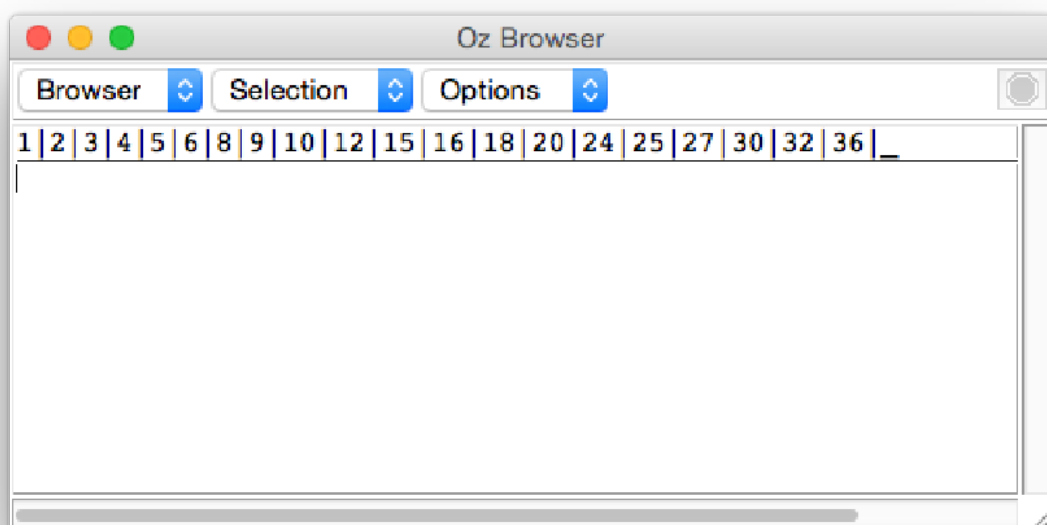
H jest nieskończonym i
uporządkowanym strumieniem
liczb postaci $2^a 3^b 5^c$

```
proc {Touch N H}
  if N>0 then {Touch N-1 H.2} else skip end
end
```

↖
ogon listy H

```
{Touch 20 H}
```

Współbieżność deklaratywna



Współbieżność z przesyłaniem komunikatów

- Przesyłanie komunikatów
- Abstrakcja tworzenia portu
- Przykład

Współbieżność z przesyłaniem komunikatów

Język modelowy

```
<S> ::=  
  skip  
  | <S>1 <S>2  
  | local <X> in <S> end  
  | <X>1=<X>2  
  | <X>=<V>  
  | if <X> then <S>1 else <S>2 end  
  | case <X> of <pattern> then <S>1 else <S>2 end  
  | {<X> <y>1 ... <y>n}  
  | thread <S> end  
  | {NewPort <y> <x>}  
  | {Send <x> <y>}
```

Współbieżność z przesyłaniem komunikatów

Operacje tworzenia kanału i wysyłania do niego:

- `{NewPort S P}` utworzenie nowego portu z punktem wejścia P i strumieniem S.
- `{Send P X}` asynchroniczne wysłanie X do strumienia odpowiadającego punktowi wejścia P.

declare S P in

`{NewPort S P}`

`{Browse S}`

`{Send P a}`

`{Send P b}`

zmiany w Browserze

`S<future>`

`al_<future>`

`abl_<future>`

Współbieżność z przesyłaniem komunikatów

Abstrakcja tworzenia portu

fun {NewPortObject Init Fun}

Sin Sout **in**

thread {FoldL Sin Fun Init Sout} **end**

{NewPort Sin}

end

- Init — stan początkowy
- Fun — funkcja zmiany stanu

$\text{Fun} : \text{STAN} \times \text{KOMUNIKAT} \mapsto \text{STAN}$

Współbieżność z przesyłaniem komunikatów

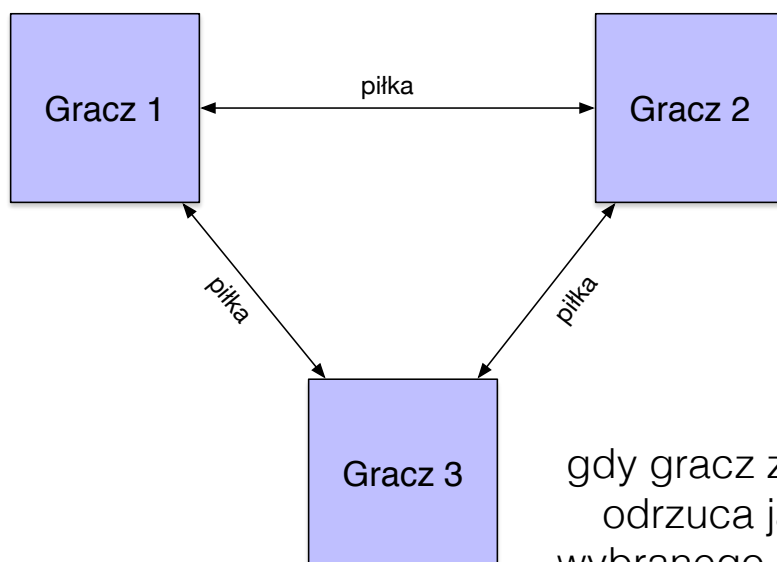
Abstrakcja tworzenia portu

```
fun {NewPortObject2 Proc}  
  Sin in  
    thread for Msg in Sin do {Proc Msg} end end  
    {NewPort Sin}  
end
```

- Proc — procedura wywoływana dla każdego komunikatu

Współbieżność z przesyłaniem komunikatów

Przykład



gdy gracz złapie piłkę, to odrzuca ją do losowo wybranego innego gracza

Współbieżność z przesyłaniem komunikatów

```
declare NewPortObject2 Player P1 P2 P3 in
fun {NewPortObject2 Proc}
  Sin in
    thread for Msg in Sin do {Proc Msg} end end
    {NewPort Sin}
end
fun {Player Others}
  {NewPortObject2
    proc {$ Msg}
      case Msg of ball then
        Ran={OS.rand} mod {Width Others} + 1
      in
        {Send Others.Ran ball}
      end
    end}
end
```

Współbieżność z przesyłaniem komunikatów

```
P1={Player others(P2 P3)}
P2={Player others(P1 P3)}
P3={Player others(P1 P2)}
```

utworzenie graczy

```
{Send P1 ball}
```

zainicjowanie gry