Addendum to Chapter 9: Direct Computation of Polynomial Representations for Sequences

Jan van Eijck (February, 2011)

In this addendum we describe a direct way to compute a polynomial representation from a sequence of numbers. Gaussian elimination is one possibility, but the so-called calculus of finite differences from combinatorics suggests another method.

```
import POL
import Ratio
```

First we introduce a new infix symbol for the operation of taking a *falling power* (or: *falling factorial*, or: *lower factorial*).

Common notation for falling powers: $x^{\underline{n}}$ (an alternative notation is $(x)_n$). $x^{\underline{n}}$ is defined as

$$x(x-1)\cdots(x-n+1).$$

For example, $x^{\underline{3}}$ equals $x(x-1)(x-2) = x^3 - 3x^2 + 2x$, and $2x^{\underline{2}}$ equals $2x(x-1) = 2x^2 - 2x$. By stipulation $x^{\underline{0}} = 1$.

```
infixr 8  ^-

(^-) :: Integral a => a -> a -> a
x ^- 0 = 1
x ^- n = (x ^- (n-1)) * (x - n + 1)
```

In a similar way, we can define *rising powers* (or: *rising factorials*, *ascending factorials*, *upper factorials*). Common notation for rising powers: $x^{\overline{n}}$. We are not going to use them below, but we throw them in for good measure.

```
infixr 8  ^+

(^+) :: Integral a => a -> a -> a
x ^+ 0 = 1
x ^+ n = (x ^+ (n-1)) * (x + n - 1)
```

We will call a polynomial representation where the exponents express falling factorials a *Newton polynomial* (Isaac Newton used these in the form of the so-called Newton interpolation formula, in his *Principia Mathematica*).

The so-called Newton series gives a way to express a polynomial f of degree n as a Newton polynomial, given that we know its list of differences for $f(0)$. Notation: $\Delta^n f$ gives the function which is the n-th difference of f. The Newton series is defined by:

$$f(x) = \sum_{k=0}^{n} \frac{\Delta^k f(0)}{k!} \cdot x^{\underline{k}}.$$

Here $k!$ is the factorial function (implemented in Haskell as `product [1..k]`), and $x^{\underline{k}}$ is a falling power. The Newton series allows us to compute a polynomial representation in terms of exponent lists from a list of first differences, but we should keep in mind that the exponents express falling powers.

Here is an example. Consider the function $f(x) = 2x^3 + 3x$. This function is of the third degree, so we compute $f(0)$, $\Delta f(0)$, $\Delta^2 f(0)$, $\Delta^3 f(0)$.

$$\begin{aligned}
f(0) &= 0 \\
\Delta f(0) &= f(1) - f(0) = 5 \\
\Delta^2 f(0) &= \Delta f(1) - \Delta f(0) \\
&= \Delta f(1) - 5 = f(2) - f(1) - 5 = 22 - 5 - 5 = 12 \\
\Delta^3 f(0) &= \Delta^2 f(1) - \Delta^2 f(0) = \Delta^2 f(1) - 12 \\
&= \Delta f(2) - \Delta f(1) - 12 = (f(3) - f(2)) - (f(2) - f(1)) - 12 \\
&= f(3) - 2f(2) + f(1) - 12 = 12.
\end{aligned}$$

We could have used `difLists` to compute the list of first differences, of course:

```
*Main> map head $ difLists [map (\x -> 2*x^3 + 3*x) [0..8]]
[12,12,5,0]
```

Newton's formula now gives:

$$\begin{aligned} f(x) &= \frac{\Delta^0 f(0)}{0!} \cdot x^{\underline{0}} + \frac{\Delta^1 f(0)}{1!} \cdot x^{\underline{1}} + \frac{\Delta^2 f(0)}{2!} \cdot x^{\underline{2}} + \frac{\Delta^3 f(0)}{3!} \cdot x^{\underline{3}} \\ &= 5x^{\underline{1}} + 6x^{\underline{2}} + 2x^{\underline{3}}. \end{aligned}$$

We assume that the differences are given in a list $[x_0, \dots, x_n]$, where $x_i = \Delta^i f(0)$. Then the implementation of the Newton series formula is as follows:

```
newton :: (Fractional a, Enum a) => [a] -> [a]
newton xs =
  [ x / product [1..fromInteger k] | (x,k) <- zip xs [0..]]
```

The list of first differences can be computed from the output of the `difLists` function, as follows:

```
firstDifs :: [Integer] -> [Integer]
firstDifs xs = reverse $ map head (difLists [xs])
```

Mapping a list of integers to a Newton polynomial representation (list of factors of the exponents, with the exponents expressing falling powers):

```
list2npol ::  [Integer] -> [Rational]
list2npol = newton . map fromInteger. firstDifs
```

This is not yet exactly what we want: we still need to map Newton falling powers to standard powers. This is a matter of applying combinatorics, by means of a conversion formula that uses the so-called *Stirling cyclic numbers*, or *Stirling numbers of the first kind*. The number $\begin{bmatrix} n \\ k \end{bmatrix}$ gives the number of ways in which a set of n elements can be partitioned into k cycles. It also gives the coefficient of x^k in the polynomial $x^{\underline{n}}$. In other words, its defining relation is:

$$x^{\underline{n}} = \sum_{k=1}^{n} \begin{bmatrix} n \\ k \end{bmatrix} (-1)^{n-k} x^k. \qquad (*)$$

Note that $(-1)^{n-k}$ takes care of the sign swaps as we step down through the powers. Looking at (*) as a definition of the coefficients of exponents in $x^{\underline{n}}$, we can work out the definition of $\begin{bmatrix} n \\ k \end{bmatrix}$, as follows. First, since $x^{\underline{0}} = 1$, the coefficient of exponent 0 in $x^{\underline{0}}$ is 1, which means that $\begin{bmatrix} 0 \\ 0 \end{bmatrix} = 1$. Since the coefficient of any exponent $k > 0$ in $x^{\underline{0}}$ equals 0, we have $\begin{bmatrix} 0 \\ k \end{bmatrix} = 0$ for $k > 0$. (There is one way to arrange the empty set in 0-sized cycles, and there are no ways to arrange the empty set in k-sized cycles for $k > 0$.)

Notice that it follows from the definition of falling powers that:

$$x^{\underline{n}} = x^{\underline{n-1}} \cdot (x - n + 1) = x x^{\underline{n-1}} - (n-1) x^{\underline{n-1}}$$

Therefore, the following holds for the coefficient of exponent x^k in $x^{\underline{n}}$ (this coefficient is given by $\begin{bmatrix} n \\ k \end{bmatrix}$):

$$\begin{bmatrix} n \\ k \end{bmatrix} = \begin{bmatrix} (n-1) \\ (k-1) \end{bmatrix} + (n-1) \begin{bmatrix} (n-1) \\ k \end{bmatrix}.$$

Notice the $(k-1)$ caused by the fact that the coefficient of exponent x^k in $x x^{\underline{n-1}}$ equals the coefficient of x^{k-1} in $x^{\underline{n-1}}$, and notice the $+$ caused by the sign swap. So we see that $\begin{bmatrix} n \\ k \end{bmatrix}$ is defined by:

$$\begin{aligned}
\begin{bmatrix} 0 \\ 0 \end{bmatrix} &:= 1 \\
\begin{bmatrix} 0 \\ k \end{bmatrix} &:= 0 \text{ for } k > 0 \\
\begin{bmatrix} n \\ k \end{bmatrix} &:= (n-1) \begin{bmatrix} (n-1) \\ k \end{bmatrix} + \begin{bmatrix} (n-1) \\ (k-1) \end{bmatrix} \text{ for } n, k > 0
\end{aligned}$$

Here is the implementation:

```
stirlingC :: Integer -> Integer -> Integer
stirlingC 0 0 = 1
stirlingC 0 _ = 0
stirlingC n k = (n-1) * (stirlingC (n-1) k)
                 + stirlingC (n-1) (k-1)
```

This definition can be used to convert from falling powers to standard powers. The implementation gives the coefficients of the map $\lambda n \cdot \sum_{k=1}^{n} \begin{bmatrix} n \\ k \end{bmatrix} x^k$.

```
fall2pol :: Integer -> [Integer]
fall2pol 0 = [1]
fall2pol n =
   0 : [ (stirlingC n k) * (-1)^(n-k) | k <- [1..n] ]
```

Next, we use this to convert Newton polynomials to standard polynomials (in coefficient list representation):

```
npol2pol :: Num a => [a] -> [a]
npol2pol xs =
  sum [ [x] * (map fromInteger $ fall2pol k) |
                          (x,k) <- zip xs [0..] ]
```

Finally, here is the function for computing a polynomial from a sequence: just a matter of composing `list2npol` with `npol2pol`.

```
list2pol :: [Integer] -> [Rational]
list2pol = npol2pol . list2npol
```

A standard application of this is so-called *curve fitting*: given a list of measurements $(0, x_0)$, $(1, x_1)$, $(2, x_2)$, $\ldots$, to find a polynomial that 'fits' all of them. This is exactly what `list2pol` does.

Here are some checks on the function that we implemented:

```
*Main> list2pol (map (\n -> 7*n^2+3*n-4) [0..100])
[(-4) % 1,3 % 1,7 % 1]
*Main> list2pol [0,1,5,14,30]
[0 % 1,1 % 6,1 % 2,1 % 3]
*Main> map (p2fct $ list2pol [0,1,5,14,30]) [0..8]
[0 % 1,1 % 1,5 % 1,14 % 1,30 % 1,55 % 1,91 % 1,140 % 1,204 % 1]
```

```
difference :: (Num a,Num b) => (a -> b) -> a -> b
difference f x = f (x+1) - f x
```

```
firstDfs :: (Num a,Num b) => (a -> b) -> [b]
firstDfs f = f 0 : firstDfs (difference f)
```

See [GKP89] or [Ros00] for (lots of) further information.

References

[GKP89] R.L. Graham, D.E. Knuth, and O. Patashnik. *Concrete Mathematics*. Addison Wesley, Reading, Mass, 1989.

[Ros00] Kenneth H. Rosen, editor. *Handbook of Discrete and Combinatorial Mathematics*. CRC Press, 2000.