Lecture 12
(with solutions)
Summer 2001
Victor Adamchik

Patterns
The whole idea of pattern matching is to select (and thus manipulate) subexpressions on the basis of their form and/or contents. Appropriate use of patterns allows us to write elegant and fast programs. However regarding the word "fast", I have to say that that is not always a case - the use of pattern matching in very large expressions may require a lot of time.

We have already used patterns in functions, for example the definition of the Laplace transform

\[ f[expr_, x_, s_] := \int_0^\infty e^{-xs} expr \, dx \]

\[ f[\sqrt{x}, x, s] \]

\[ \text{If} [\text{Re}[s] > 0, \frac{\sqrt{\pi}}{2 s^{3/2}}, \int_0^\infty e^{-s x} \sqrt{x} \, dx] \]

Here is a list of typical possibilities for patterns:
- \( x_\_ \) stands for an object named \( x \);
- \( x_\_HEAD \) stands for an object with head \( \text{HEAD} \) named \( x \);
- \( x\__ \) stands for a sequence of objects (more than one) named \( x \);
- \( x\___ \) stands for zero or more object named \( x \).

Double or triple blank represents the situation when you don't know (or don't care) of the number of arguments.

Consider the following declaration

\[ g[x_, \___] := \text{If}[x = 0, x, 1] \]

\[ g[1, 2, 3, 4, 5] \]

1

We use patterns with the head to distinguish different inputs

\[ h[x_{\text{Complex}}] := x \ast \text{Conjugate}[x] \]
Often we want to define functions with quite restrictive conditions. Here is an example of alternative (OR) pattern (see Alternatives)

\[
R[x_\text{Integer}\mid x_\text{Rational}] := x + 1/x
\]
\[
\{R[2], R[2/3], R[2.3]\}
\]
\[
\{\frac{5}{2}, \frac{13}{6}, R[2.3]\}
\]

An example of test (AND) pattern (see PatternTest)

\[
Q[x_\text{Integer}\?\text{Positive}] := x - 1/x
\]
\[
\{Q[2], Q[-2], Q[2.3]\}
\]
\[
\{\frac{3}{2}, Q[-2], Q[2.3]\}
\]

As the test function, we can use a pure function

\[
\text{Clear}[f];
\]
\[
f[x_\text{Integer}\?((\# > 2) \&)] := x - 2
\]
\[
\{f[2], f[4]\}
\]
\[
\{f[2], 2\}
\]

Here is another example

\[
\text{Clear}[f];
\]
\[
f[x_\?((\text{Length}[\#] \leq 1) \&)] := \text{Last}[x]
\]
\[
f[x_\?((\text{Length}[\#] > 1) \&)] := \text{First}[x]
\]
\[
\{f[[10]], f[[1, 2]]\}
\]
\[
\{10, 1\}
\]

Sometimes it is difficult to specify patterns using the above tools. As an alternative we have Condition (which is represented by /;) (see Condition)
Clear[f];
f[x_/; 1 < x < 2] := x

{f[0.5], f[1.5]}

{f[0.5], 1.5}

If you deal with the function of more than one argument, the condition should be applied to the whole expression. In the following example, there are only two possibilities for the condition:

Clear[f];
f[x_, y_/; y > x] := x + y

{f[1, 2], f[2, 1]}

{5, f[2, 1]}

Clear[f];
f[x_, y_] := x^2 + y^2

{f[1, 2], f[2, 1]}

{5, f[2, 1]}

Problem 1.
What outputs correspond to the following inputs?

Clear[f, x, y];
f[x_/NonNegative | x_Complex, y_/y > 0] := x + y

A) f[1, 2]
B) f[2, 0]

Clear[f, x, y];
f[x_/NonNegative | x_Complex, y_/y > 0] := x + y

f[I, 2]
2 + i

f[2, 0]

Problem 2.
Construct as many patterns as you can to match integers between 1 and 9.
f[x_Integer /; 1 < x < 9] := x
f[x_Integer] /; 1 < x < 9 := x
f[x_Integer] := x /; 1 < x < 9
f[x_Integer?((1 < # < 9) &)] := x
f[x_Integer?((# > 1) &)] := x /; x < 9

Clear[f]; f[x_Integer?((1 < # < 9) &)] := x

f[4]

Problem 3.
Write a function which matches polynomials with only two additive terms.

Clear[f];
f[a_ + b_] := True /; Head[a] =!= Plus && Head[b] =!= Plus
f /@ {4 x^2, 1 + 2 x^3, d x, x + x^3 + x^4}
{f[4 x^2], True, f[d x], f[x + x^3 + x^4]}

Problem 4.
What outputs correspond to the following inputs?

Clear[f, x]
f[x___] := x + 1

A) f[]
B) f[1, 2, 3]

Clear[f]; f[x___] := x + 1;
Sequence[] + 1
1

Sequence[1, 2, 3] + 1
7

List Manipulation
A variety of operations can be performed on lists. You know Take and Select:
Here is another function `Drop`, which is designed to remove particular elements from the list. We remove the first two elements

\[
\text{Drop[{a, b, c}, 2]}
\]

\[
\{c\}
\]

we remove the second elements

\[
\text{Drop[{a, b, c}, {2}]}\]

\[
\{a, c\}
\]

we remove two elements from the end

\[
\text{Drop[{a, b, c}, -2]}\]

\[
\{a\}
\]

we remove each third element

\[
\text{Drop[Range[20], {3, 20, 3}]}\]

\[
\{1, 2, 4, 5, 7, 8, 10, 11, 13, 14, 16, 17, 19, 20\}
\]

The function `Delete` is very close to `Drop`:

\[
\text{Delete[{a, b, c, d}, 2]}\]

\[
\{a, c, d\}
\]

However, `Delete` has more functionality. Here we delete second and forth elements

\[
\text{Delete[{a, b, c, d, e}, {{2}, {4}}]}\]

\[
\{a, c, e\}
\]

Another yet and very useful function is `DeleteCases`, which accepts as the second argument a pattern
Problem 4.
Write a function which deletes each \(k\)-th element from a given list of the length \(n\). Then write a function which deletes each second element of this list. From the resulting list it deletes each third element, then each fourth element, and so on until it’s possible to delete. Surely, the process is terminated (when the length of the list is less than a number \(k\)). Approximate the length of the final list as a function of \(n\). An example, we start with a list of 5 element
\[
\{1,2,3,4,5\}
\]
\[
\{1,3,5\}
\]
\[
\{1,3\}
\]
\[
delete[l\_List, k\_] := \text{Drop}[l, \{k, \text{Length}[l], k\}] \quad \text{if} \quad \text{Length}[l] \geq k
\]
\[
delete[l\_List, k\_] := 1 \quad \text{if} \quad \text{Length}[l] < k
\]
\[
delete[\text{Range}[10], 2]
\]
\[
\{1, 3, 5, 7, 9\}
\]
\[
n = 5; \text{FoldList}[\text{delete}[\#1, \#2 \&, \text{Range}[n], \text{Rest}[\text{Range}[n]]]]
\]
\[
\{\{1, 2, 3, 4, 5\}, \{1, 3, 5\}, \{1, 3\}, \{1, 3\}, \{1, 3\}\}
\]
\[
\text{Clear}[f];
\]
\[
f[n\_] := \text{Length}[\text{Fold}[\text{delete}[\#1, \#2 \&, \text{Range}[n], \text{Rest}[\text{Range}[n]]]]]
\]
tab = Table[f[n], {n, 100, 10000, 100}];
ListPlot[tab, PlotJoined -> True]

- Graphics -

Make guess $f(n) \rightarrow \frac{c}{\sqrt{n}}$.

To make sure that our guess is right we do

\[ \text{tab} = \text{Table}\left[\frac{f[n]}{\sqrt{n}}, \{n, 1000, 10000, 100\}\right]; \]
\[ \text{ListPlot}\left[\text{tab}, \text{PlotJoined} \rightarrow \text{True}\right] \]

- Graphics -

Indeed, our guess is right, we need to find that constant which is between 1.12 and 1.13
\textbf{ListPlot[4/\text{tab}^2, \text{PlotJoined} \rightarrow \text{True}]}

- Graphics -

The constant is \(\frac{2}{\sqrt{\pi}}\).

\(f[20000]\)

159

\(2/\text{Sqrt[Pi]} \text{Sqrt[20000.]}\)

159.577