Chapter 4
Patterns and Substitutions

Substitution is the most fundamental operation in *Mathematica*. Its left-hand side is a pattern. In a given expression, all subexpressions matching the pattern are found and replaced by the right-hand side of the substitution.

4.1 Simple Patterns

$f[x]$ with a specific argument x.

```
In[1] := {f[x], f[y]} /. f[x] -> x^2
```

$\text{Out}[1] = \{x^2, f[y]\}$

f with an arbitrary argument.

```
In[2] := {f[x], f[y]} /. f[x_] -> x^2
```

$\text{Out}[2] = \{x^2, y^2\}$

f with two identical (arbitrary) arguments.

```
In[3] := {f[x, x], f[x, y]} /. f[x_, x_] -> g[x]
```

$\text{Out}[3] = \{g[x], f[x, y]\}$

An example of a more complicated pattern.

```
In[4] := f[g[f[x], y], h[f[x]]] /. f[g[x_, y], h[x_]] -> F[x, y]
```

$\text{Out}[4] = F[f[x], y]$

f with an argument being an arbitrary integer number.

```
In[5] := {f[x], f[2]} /. f[x_Integer] -> x^2
```

$\text{Out}[5] = \{f[x], 4\}$

In fact, such a form of an arbitrary argument checks its head. This substitution applies when the argument's head is g.

```
In[6] := {f[g[x, y]], f[h[x, y]]} /. f[x_g] -> x^2
```

$\text{Out}[6] = \{g[x, y]^2, f[h[x, y]]\}$

And this one—when the argument is a sum.

```
In[7] := {f[{x, y}], f[x + y]} /. f[x_Plus] -> x^2
```

$\text{Out}[7] = \{f[\{x, y\}], (x + y)^2\}$

A. Grozin, *Introduction to Mathematica® for Physicists*, Graduate Texts in Physics,
DOI 10.1007/978-3-319-00894-3_4, © Springer International Publishing Switzerland 2014

And this one—when the argument is a list. By the way, note what happens when a list is being squared.

In[8] := {f[{x,y}],f[x+y]}/.f[x_List]->x^2

Out[8] = $\{\{x^2, y^2\}, f[x+y]\}$

One more example.

In[9] := a = Sqrt[x]/Sqrt[y]

Out[9] = $\frac{\sqrt{x}}{\sqrt{y}}$

In[10] := a/.{Sqrt[x]->u,Sqrt[y]->v}

Out[10] = $\frac{u}{\sqrt{y}}$

Why hasn't the second substitution triggered?

In[11] := FullForm[a]

Out[11]//FullForm =

Times[Power[x,Rational[1,2]],Power[y,Rational[-1,2]]]

a does not contain $y^{1/2}$, only $y^{-1/2}$; therefore, the substitution $y^{1/2} \to v$ does not work.

Out[11] = **Clear**[a]

4.2 One-Shot and Repeated Substitutions

Here is an expression.

In[12] := a = x^2+y^2

Out[12] = $x^2 + y^2$

Let's increase x by 1 in it. This example demonstrates that a substitution is not applied repeatedly. *Mathematica* searches for subexpressions matching the pattern (in this case x) in the expression a. After finding such a subexpression, *Mathematica* replaces it by the right-hand side. The result of such a replacement is not searched again for subexpressions matching the pattern.

In[13] := a = a/.x->x+1

Out[13] = $(1+x)^2 + y^2$

A list of substitutions can be applied to an expression. They are all applied in parallel—*Mathematica* searches for subexpressions matching some pattern from the list and replaces these subexpressions by the corresponding right-hand side. Therefore two symbols can be interchanged in an expression in this simple way.

In[14] := a = a/.{x->y,y->x}

Out[14] = $x^2 + (1+y)^2$

In[15] := Clear[a]

The operator //. (in contrast to /.) applies a substitution repeatedly, while it is applicable. If several substitutions are applicable to some subexpression, *Mathematica* first applies the most specific one (it is not always easy to determine which substitution is more specific and which is more general; in simple cases, this is clear).

In[16] := fac[10]//.{fac[0]->1,fac[n_]->n*fac[n-1]}
Out[16] = 3628800
By the way, what are the real names of /. and //. ?
In[17] := FullForm[Hold[a/.x->y]]
Out[17]//FullForm =
    Hold[ReplaceAll[a, Rule[x, y]]]
In[18] := FullForm[Hold[a//.x->y]]
Out[18]//FullForm =
    Hold[ReplaceRepeated[a, Rule[x, y]]]

4.3 Products

Let's take a product.
In[19] := FullForm[a = 2*x*y*z]
Out[19]//FullForm =
    Times[2, x, y, z]
The pattern xy is considered contained in this product, though the internal representation of a does not contain Times$[x, y]$ explicitly.
In[20] := a/.x*y->z
Out[20] = $2z^2$
And this product does not contain xy.
In[21] := FullForm[a = 2*x^2*y*z]
Out[21]//FullForm =
    Times[2, Power[x, 2], y, z]
In[22] := a/.x*y->z
Out[22] = $2x^2yz$
This product contains powers of x and y.
In[23] := FullForm[a = 2*x^2*y^3*z]
Out[23]//FullForm =
    Times[2, Power[x, 2], Power[y, 3], z]
We want to replace each product of powers of x and y by the function f of these powers. Such a problem occurs very often. For example, we want to integrate some class of expressions, and we know the result of integration as a function of powers of some variables (or subexpressions).
In[24] := a/.x^n_*y^m_->f[n,m]
Out[24] = $2zf[2, 3]$
This works OK. And here?
In[25] := FullForm[a = 2*x^2*y*z]
Out[25]//FullForm =
    Times[2, Power[x, 2], y, z]
In[26] := a/.x^n_*y^m_->f[n,m]
Out[26] = $2x^2yz$

This doesn't work. The product a does not contain a product of *powers* of x and y: the symbol y is not in the argument of the function Power. In the next example the substitution works again—as we have seen, *Mathematica* considers dividing by y as multiplying by y^{-1}.

In[27] := FullForm[a = 2 * x^2 * z/y]

Out[27]//FullForm =

Times[2, Power[x, 2], Power[y, −1], z]

In[28] := a/.x^n_*y^m_−>f[n,m]

Out[28] = $2zf[2,-1]$

Let's return to the previous expression. How can we instruct *Mathematica* to consider y as a particular case of the pattern "y to an arbitrary power"? This is what an optional arbitrary argument $m\_.$ is for. When it is used in an exponent, its default value (which is used when there is no power at all) is 1.

In[29] := FullForm[a = 2 * x^2 * y * z]

Out[29]//FullForm =

Times[2, Power[x, 2], y, z]

In[30] := a/.x^n_.*y^m_.−>f[n,m]

Out[30] = $2zf[2,1]$

In[31] := FullForm[a = 2 * x * y * z]

Out[31]//FullForm =

Times[2, x, y, z]

In[32] := a/.x^n_.*y^m_.−>f[n,m]

Out[32] = $2zf[1,1]$

So far so good. But what if the symbol y is absent? Will *Mathematica* consider this as a particular case of the pattern "y to an arbitrary power" with the power equal 0? It will not.

In[33] := FullForm[a = 2 * x^2 * z]

Out[33]//FullForm =

Times[2, Power[x, 2], z]

In[34] := a/.x^n_.*y^m_.−>f[n,m]

Out[34] = $2x^2z$

The following method will work always. Let's collect several test expressions to a list.

In[35] := a = {2 * x * y * z, 2 * x^2 * y * z, 2 * x^2 * y^3 * z, 2 * x^2 * z/y, 2 * x^2 * z, 2 * z}

Out[35] = $\left\{2xyz, 2x^2yz, 2x^2y^3z, \frac{2x^2z}{y}, 2x^2z, 2z\right\}$

The method is as follows. Multiply our expression by $f[0,0]$, and apply a list of substitutions. If f with some arguments is multiplied by an arbitrary power of x, then the first argument of f is increased by this power. Of course, if x is not raised to any power, we want to use the default power equal to 1. Powers of y are treated in the same way. We need to use the repeated substitution //.—after one substitution from the list has been applied (e.g., the one about x), we want the other one to be applied to the result (to take y into account).

In[36] := $s = \{x\text{^}l\_.*f[n\_,m\_]\text{->}f[n+l,m], y\text{^}l\_.*f[n\_,m\_]\text{->}f[n,m+l]\}$
Out[36] = $\left\{x^{l\_.} f[n\_,m\_] \to f[l+n,m], y^{l\_.} f[n\_,m\_] \to f[n,l+m]\right\}$
In[37] := $a*f[0,0]//.s$
Out[37] = $\{2zf[1,1], 2zf[2,1], 2zf[2,3], 2zf[2,-1], 2zf[2,0], 2zf[0,0]\}$
In[38] := Clear[a,s]

4.4 Sums

Substitutions for sums are similar to those for products. They are used much more rarely. Don't use them if you can avoid this.
In[39] := FullForm[$a = x+y+z+2$]
Out[39]//FullForm =
Plus[2, x, y, z]
In[40] := $a/.x+y\text{->}z$
Out[40] = $2+2z$
In[41] := FullForm[$a = 2*x+y+z+2$]
Out[41]//FullForm =
Plus[2, Times[2, x], y, z]
In[42] := $a/.x+y\text{->}z$
Out[42] = $2+2x+y+z$
This substitution replaces a sum of x and y with arbitrary coefficients by the function f of these coefficients.
In[43] := FullForm[$a = 2*x+3*y+z+2$]
Out[43]//FullForm =
Plus[2, Times[2, x], Times[3, y], z]
In[44] := $a/.n\_*x+m\_*y\text{->}f[n,m]$
Out[44] = $2+z+f[2,3]$
In[45] := FullForm[$a = 2*x+y+z+2$]
Out[45]//FullForm =
Plus[2, Times[2, x], y, z]
In[46] := $a/.n\_*x+m\_*y\text{->}f[n,m]$
Out[46] = $2+2x+y+z$
In[47] := FullForm[$a = 2*x-y+z+2$]
Out[47]//FullForm =
Plus[2, Times[2, x], Times[-1, y], z]
In[48] := $a/.n\_*x+m\_*y\text{->}f[n,m]$
Out[48] = $2+z+f[2,-1]$
Here again an optional arbitrary argument can be used. When it is used as a factor, a subexpression is considered matching this pattern even if there is no such a factor, and its value in this case is taken to be 1.

In[49] := FullForm[a = 2 * x + y + z + 2]
Out[49]//FullForm =
    Plus[2, Times[2, x], y, z]
In[50] := a/.n_.*x+m_.*y->f[n,m]
Out[50] = 2 + z + f[2, 1]
In[51] := FullForm[a = x + y + z + 2]
Out[51]//FullForm =
    Plus[2, x, y, z]
In[52] := a/.n_.*x+m_.*y->f[n,m]
Out[52] = 2 + z + f[1, 1]
In[53] := FullForm[a = x + z + 2]
Out[53]//FullForm =
    Plus[2, x, z]
In[54] := a/.n_.*x+m_.*y->f[n,m]
Out[54] = 2 + x + z
And here is our method which always works.
In[55] := a = {x+y+z+2, 2*x+y+z+2, 2*x+3*y+z+2, 2*x-y+z+2,
x+z+2, z+2}
Out[55] = {2 + x + y + z, 2 + 2x + y + z, 2 + 2x + 3y + z, 2 + 2x − y + z, 2 + x + z, 2 + z}
In[56] := s = {l_.*x + f[n_, m_] -> f[n + l, m], l_.*y + f[n_, m_] -> f[n, m + l]}
Out[56] = {f[n_, m_] + x l_. → f[l + n, m], f[n_, m_] + y l_. → f[n, l + m]}
In[57] := a + f[0, 0]//.s
Out[57] = {2 + z + f[1, 1], 2 + z + f[2, 1], 2 + z + f[2, 3], 2 + z + f[2, −1],
    2 + z + f[1, 0], 2 + z + f[0, 0]}
In[58] := Clear[a, s]

4.5 Conditions

Substitutions which apply only when an arbitrary variable satisfies some condition are often needed.
In[59] := {f[1.5], f[3/2], f[x/2]}/.f[x_?NumberQ]->x^2
Out[59] = $\left\{2.25, \frac{9}{4}, f\left[\frac{x}{2}\right]\right\}$
But this method is not very general. It checks a condition depending on a single variable. The operator /; can be applied to a pattern (or its part). It can be read as "such that." The condition in it can depend on several arbitrary variables.
In[60] := s = {fac[0]->1, fac[n_Integer/;n > 0]->n * fac[n − 1]}
Out[60] = {fac[0] → 1, fac[n_Integer/; n > 0] → n fac[−1 + n]}
In[61] := {fac[10], fac[−10]}//.s
Out[61] = {3628800, fac[−10]}
Internally, this operator is the function Condition.
In[62] := FullForm[s]
Out[62]//FullForm =
    List[Rule[fac[0], 1],

```
        Rule[fac[Condition[Pattern[n, Blank[Integer]], Greater[n, 0]]],
            Times[n, fac[Plus[-1, n]]]]]
```

In[63] := Clear[s]

One common case is when you want to replace $f[x]$ by $g[x]$ only for some values of x.

In[64] := f[a] + f[b] + f[c] /. f[x_] /; x == a || x == b -> g[x]

Out[64] = $f[c] + g[a] + g[b]$

4.6 Variable Number of Arguments

A pattern can involve a construct which matches not a single subexpression but an arbitrary-length subsequence of arguments of a function. This is very convenient for working with functions having an arbitrary number of arguments. Let's consider an example. The function f has any number of arguments. We want to shuffle them in the opposite order. First, let's put a fence at the end of the argument list.

In[65] := a = f[x, y, z]

Out[65] = $f[x, y, z]$

In[66] := a = a /. f[x___] -> f[x, Fence]

Out[66] = $f[x, y, z, \text{Fence}]$

Now we take the arguments from the left one by one and throw them over the fence (placing them immediately after the fence).

In[67] := a = a // . f[x_, y___, Fence, z___] -> f[y, Fence, x, z]

Out[67] = $f[\text{Fence}, z, y, x]$

Now the fence is at the left, and the arguments are after it in the opposite order. What's left is to remove the fence.

In[68] := a = a /. f[Fence, x___] -> f[x]

Out[68] = $f[z, y, x]$

Of course, this method only works when the symbol Fence is not present among the arguments. Here is the method which always works. Let the part of the arguments which has not yet been processed be in the first list and the processed part—in the second one. We take the arguments one by one from the beginning of the first list and move them to the beginning of the second one.

In[69] := a = a /. f[x___] -> f[{x}, {}]

Out[69] = $f[\{z, y, x\}, \{\}]$

In[70] := a = a // . f[{x_, y___}, {z___}] -> f[{y}, {x, z}]

Out[70] = $f[\{\}, \{x, y, z\}]$

In[71] := a = a /. f[{}, {x___}] -> f[x]

Out[71] = $f[x, y, z]$

In addition to $x\_\_\_$ (with three underscores), which means an arbitrary subsequence of arguments of a function (maybe an empty one), there is also $x\_\_$ (with two underscores)—an arbitrary nonempty subsequence of arguments. I find the first construct more useful.