Expert Systems

In technical terms, Mathematica is an **Expert System**. Knowledge is added in form of **Transformation Rules**. An expression is transformed until no more rules apply.

**Example:**

```math
myAbs[x_] := x /; NonNegative[x]
myAbs[x_] := -x /; Negative[x]
```

We get:

```math
myAbs[3] ➞ 3
myAbs[-5] ➞ 5
myAbs[2 + 3 I] ➞ myAbs[2 + 3 I]
   – no rule for complex arguments so far
myAbs[x] ➞ myAbs[x]
   – no match either
```
Immediate and Delayed Assignment

Transformations can either be

- added “permanently” in form of Definitions,
  \[
  \text{norm}[\text{vec}_\_] := \text{Sqrt}[\text{vec} . \text{vec}]
  \]
  \[
  \text{norm}[\{1, 0, 2\}] \rightarrow \text{Sqrt}[5]
  \]
- applied once using Rules:
  \[
  a + b + c \/. a \rightarrow 2 c \rightarrow b + 3 c
  \]

Transformations can be Immediate or Delayed. Consider:

\[
\{r, r\} \/. r \rightarrow \text{Random[]} \rightarrow \{0.823919, 0.823919\}
\]
\[
\{r, r\} \/. r :> \text{Random[]} \rightarrow \{0.356028, 0.100983\}
\]

Mathematica is one of those programs, like \TeX, where you wish you'd gotten a US keyboard for all those braces and brackets.

T. Hahn, Introduction to Mathematica – p.3
Almost everything is a List

All Mathematica objects are either Atomic, e.g.

\[
\text{Head}[133] \Rightarrow \text{Integer}
\]
\[
\text{Head}[a] \Rightarrow \text{Symbol}
\]

or (generalized) Lists with a Head and Elements:

\[
\text{expr} = a + b
\]
\[
\text{FullForm}[\text{expr}] \Rightarrow \text{Plus}[a, b]
\]
\[
\text{Head}[\text{expr}] \Rightarrow \text{Plus}
\]
\[
\text{expr}[[0]] \Rightarrow \text{Plus} \quad \text{— same as Head[expr]}
\]
\[
\text{expr}[[1]] \Rightarrow a
\]
\[
\text{expr}[[2]] \Rightarrow b
\]
List-oriented Programming

Using Mathematica’s list-oriented commands is almost always of advantage in both speed and elegance.

Consider:

array = Table[Random[], {10^7}];

test1 := Block[{sum = 0},
  Do[sum += array[[i]], {i, Length[array]}];
  sum]

test2 := Apply[Plus, array]

Here are the timings:

Timing[test1][[1]] ➯ 31.63 Second
Timing[test2][[1]] ➯ 3.04 Second
Map, Apply, and Pure Functions

**Map** applies a function to all elements of a list:

\[
\text{Map}[f, \{a, b, c\}] \rightarrow \{f[a], f[b], f[c]\}
\]

\[f \text{ /@ } \{a, b, c\} \rightarrow \{f[a], f[b], f[c]\} \quad \text{— short form}\]

**Apply** exchanges the head of a list:

\[
\text{Apply}[\text{Plus}, \{a, b, c\}] \rightarrow a + b + c
\]

\[\text{Plus } @@ \{a, b, c\} \rightarrow a + b + c \quad \text{— short form}\]

**Pure Functions** are a concept from formal logic. A pure function is defined ‘on the fly’:

\[
(# + 1)& \text{ /@ } \{4, 8\} \rightarrow \{5, 9\}
\]

The \# (same as \#1) represents the first argument, and the \& defines everything to its left as the pure function.
List Operations

**Flatten** removes all sub-lists:

\[
\text{Flatten}[f[x, f[y], f[f[z]]]] \rightarrow f[x, y, z]
\]

**Sort** and **Union** sort a list. **Union** also removes duplicates:

\[
\text{Sort}[[3, 10, 1, 8]] \rightarrow \{1, 3, 8, 10\}
\]
\[
\text{Union}[[c, c, a, b, a]] \rightarrow \{a, b, c\}
\]

**Prepend** and **Append** add elements at the front or back:

\[
\text{Prepend}[r[a, b], c] \rightarrow r[c, a, b]
\]
\[
\text{Append}[r[a, b], c] \rightarrow r[a, b, c]
\]

**Insert** and **Delete** insert and delete elements:

\[
\text{Insert}[h[a, b, c], x, \{2\}] \rightarrow h[a, x, b, c]
\]
\[
\text{Delete}[h[a, b, c], \{2\}] \rightarrow h[a, c]
\]
Patterns

One of the most useful features is **Pattern Matching:**

- matches one object
- matches one or more objects
- matches zero or more objects
- named pattern (for use on the r.h.s.)
- pattern with head h
- default value
- conditional pattern
- conditional pattern

Patterns take function overloading to the limit, i.e. functions behave differently depending on *details* of their arguments:

```mathematica
Attributes[Pair] = {Orderless}
Pair[p_Plus, j_] := Pair[#1, j] & /@ p
Pair[n_?NumberQ i_, j_] := n Pair[i, j]
```

T. Hahn, Introduction to Mathematica – p.8
Attributes

Attributes characterize a function’s behaviour before and while it is subjected to pattern matching. For example,

\[
\text{Attributes}[f] = \{\text{Listable}\}
\]

\[
f[l\_\text{List}] := g[l]
\]

\[
f[{1, 2}] \rightarrow \{f[1], f[2]\} \quad \text{— definition is never seen}
\]

Important attributes: Flat, Orderless, Listable, HoldAll, HoldFirst, HoldRest.

The Hold... attributes are needed to pass variables by reference:

\[
\text{Attributes}[\text{listadd}] = \{\text{HoldFirst}\}
\]

\[
\text{listadd}[x\_, \text{other}\_] := x = \text{Flatten}[\{x, \text{other}\}]
\]

This would not work if \(x\) were expanded before invoking \text{listadd}, i.e. passed by value.
Memorizing Values

For longer computations, it may be desirable to ‘remember’ values once computed. For example:

\[
\begin{align*}
\text{fib}[1] &= \text{fib}[2] = 1 \\
\text{fib}[i_] &:= \text{fib}[i] = \text{fib}[i - 2] + \text{fib}[i - 1] \\
\text{fib}[4] &\rightarrow 3 \\
?\text{fib} &\rightarrow \text{Global}`\text{fib} \\
\text{fib}[1] &= 1 \\
\text{fib}[2] &= 1 \\
\text{fib}[3] &= 2 \\
\text{fib}[4] &= 3 \\
\text{fib}[i_] &:= \text{fib}[i] = \text{fib}[i - 2] + \text{fib}[i - 1]
\end{align*}
\]

Note that Mathematica places more specific definitions before more generic ones.
Decisions

Mathematica’s If Statement has three entries: for True, for False, but also for Undecidable. For example:

\[
\begin{align*}
\text{If}[8 > 9, 1, 2] & \Rightarrow 2 \\
\text{If}[a > b, 1, 2] & \Rightarrow \text{If}[a > b, 1, 2] \\
\text{If}[a > b, 1, 2, 3] & \Rightarrow 3
\end{align*}
\]

Property-testing Functions end in Q: EvenQ, PrimeQ, NumberQ, MatchQ, OrderedQ, … These functions have no undecided state: in case of doubt they return False.

Conditional Patterns are usually faster:

\[
\begin{align*}
good[a\_, b\_] & := \text{If}[\text{TrueQ}[a > b], 1, 2] \\
& \quad \text{— TrueQ removes ambiguity} \\
better[a\_, b\_] & := 1 \;/; a > b \\
better[a\_, b\_] & = 2
\end{align*}
\]

T. Hahn, Introduction to Mathematica – p.11
Equality

Just as with decisions, there are several types of equality, decidable and undecidable:

\[
\begin{align*}
a &= b & \Rightarrow & \ a &= b \\
a &=& b & \Rightarrow & \ False \\
a &= a & \Rightarrow & \ True \\
a &=& a & \Rightarrow & \ True
\end{align*}
\]

The full name of ‘===’ is \texttt{SameQ} and works as the \texttt{Q} indicates: in case of doubt, it gives \texttt{False}. It tests for \texttt{Structural Equality}.

Of course, equations to be solved are stated with ‘==’:

\[
\text{Solve}[x^2 == 1, x] \Rightarrow \{\{x \rightarrow -1\}, \{x \rightarrow 1\}\}
\]

Needless to add, ‘=’ is a definition and quite different:

\[
x = 3 \quad \text{– assign 3 to} \ x
\]
Selecting Elements

**Select** selects elements fulfilling a criterium:

\[
\text{Select}[[1, 2, 3, 4, 5], \# > 3 \&] \rightarrow \{4, 5\}
\]

**Cases** selects elements matching a pattern:

\[
\text{Cases}[[1, a, f[x]], \_\text{Symbol}] \rightarrow \{a\}
\]

Using **Levels** is generally a very fast way to extract parts:

\[
\text{list} = \{f[x], 4, \{g[y], h\}\}
\]

\[
\text{Depth}[\text{list}] \rightarrow 4 \quad \text{— list is 4 levels deep (0, 1, 2, 3)}
\]

\[
\text{Level}[\text{list}, \{1\}] \rightarrow \{f[x], 4, \{g[y], h\}\}
\]

\[
\text{Level}[\text{list}, \{2\}] \rightarrow \{x, g[y], h\}
\]

\[
\text{Level}[\text{list}, \{3\}] \rightarrow \{y\}
\]

\[
\text{Level}[\text{list}, \{-1\}] \rightarrow \{x, 4, y, h\}
\]

\[
\text{Cases}[\text{expr}, \_\text{Symbol}, \{-1\}]//\text{Union}
\]

\quad \text{— find all variables in expr}
Mathematical Functions

Mathematica is equipped with a large set of mathematical functions, both for symbolic and numeric operations.

Some examples:

- `Integrate[x^2, {x, 3, 5}]` — integral
- `D[f[x], x]` — derivative
- `Sum[i, {i, 50}]` — sum
- `Series[Sin[x], {x, 1, 5}]` — series expansion
- `Simplify[(x^2 - x y)/x]` — simplify
- `Together[1/x + 1/y]` — put on common denominator
- `Inverse[mat]` — matrix inverse
- `Eigenvalues[mat]` — eigenvalues
- `PolyLog[2, 1/3]` — polylogarithm
- `LegendreP[11, x]` — Legendre polynomial
- `Gamma[.567]` — Gamma function
Mathematica has formidable graphics capabilities:

```
Plot[ArcTan[x], {x, 0, 2.5}]
ParametricPlot[{Sin[x], 2 Cos[x]}, {x, 0, 2 Pi}]
Plot3D[1/(x^2 + y^2), {x, -1, 1}, {y, -1, 1}]
ContourPlot[x y, {x, 0, 10}, {y, 0, 10}]
```

Output can be saved to a file with `Export`:

```
plot = Plot[Abs[Zeta[1/2 + x I]], {x, 0, 50}]
Export["zeta.eps", plot, "EPS"]
```

**Hint:** To get a high-quality plot with proper \LaTeX\ labels, don’t waste your time fiddling with the `Plot` options. Use the `psfrag` \LaTeX\ package.
Numerics

Mathematica can express Exact Numbers, e.g.
\[ \sqrt{2}, \pi, \frac{27}{4} \]

It can also do Arbitrary-precision Arithmetic, e.g.
\[ N[Erf[28/33], 25] \Rightarrow 0.7698368826185349656257148 \]

But: Exact or arbitrary-precision arithmetic is fairly slow!
Mathematica uses Machine-precision Reals for fast arithmetic.
\[ N[Erf[28/33]] \Rightarrow 0.769836882618535 \]

Arrays of machine-precision reals are internally stored as Packed Arrays (this is invisible to the user) and in this form attain speeds close to compiled languages on certain operations, e.g. eigenvalues of a large matrix.

T. Hahn, Introduction to Mathematica – p.16
Compiled Functions

Mathematica can ‘compile’ certain functions for efficiency.
This is not compilation into assembler language, but rather a strong typing of an expression such that intermediate data types do not have to be determined dynamically.

```
fun[x_] := Exp[-((x - 3)^2/5)]
cfun = Compile[{x}, Exp[-((x - 3)^2/5)]]
time[f_] := Timing[Table[f[1.2], {10^5}]][[1]]
time[fun] ↩ 2.4 Second
time[cfun] ↩ 0.43 Second
```

Compile is implicit in many numerical functions, e.g. in Plot.

In a similar manner, Dispatch hashes long lists of rules beforehand, to make the actual substitution faster.
Blocks and Modules

Block implements Dynamical Scoping
A local variable is known everywhere, but only for as long as the block executes ("temporal localization").

Module implements Lexical Scoping
A local variable is known only in the block it is defined in ("spatial localization"). This is how scoping works in most high-level languages.

```
printa := Print[a]
a = 7
btest := Block[{a = 5}, printa]
mtest := Module[{a = 5}, printa]
```

```
btest ☞ 5
mtest ☞ 7
```
DownValues and UpValues

Definitions are usually assigned to the symbol being defined: this is called **DownValue**.

For seldomly used definitions, it is better to assign the definition to the next lower level: this is an **UpValue**.

```
x/: Plus[x, y] = z
```

```
?x
```

```
Global`x
```

```
x/: x + y = z
```

This is better than assigning to **Plus directly, because Plus is a very common operation.**

In other words, Mathematica “looks” one level inside each object when working off transformations.
Mathematica knows some functions to be **Output Forms**. These are used to format output, but don’t “stick” to the result:

\[
\{\{1, 2\}, \{3, 4\}\}\text{\color{red}{\text{//MatrixForm}}}
\]

\[
\begin{pmatrix}
1 & 2 \\
3 & 4
\end{pmatrix}
\]

\text{Head[\%]} \quad \Rightarrow \quad \text{List} \quad \text{— not MatrixForm}

**Some important output forms:**

\text{InputForm, FullForm, Shallow, MatrixForm, TableForm, TeXForm, CForm, FortranForm.}

\text{TeXForm[alpha/(4 Pi)]} \quad \Rightarrow \quad \frac{\alpha}{4\pi}

\text{CForm[alpha/(4 Pi)]} \quad \Rightarrow \quad \alpha/(4.*\Pi)

\text{FullForm[alpha/(4 Pi)]}

\Rightarrow \times[\text{Rational}[1, 4], \alpha, \text{Power}[\Pi, -1]]
MathLink

The **MathLink API** connects Mathematica with external C/C++ programs (and vice versa). **J/Link** does the same for Java.

:Begin:
:Function: copysign
:Pattern: CopySign[x_?NumberQ, s_?NumberQ]
:Arguments: {N[x], N[s]}
:ArgumentTypes: {Real, Real}
:ReturnType: Real
:End:

```c
#include "mathlink.h"

double copysign(double x, double s) {
    return (s < 0) ? -fabs(x) : fabs(x);
}

int main(int argc, char **argv) {
    return MLMain(argc, argv);
}
```

**In-depth tutorial:** [http://library.wolfram.com/infocenter/TechNotes/174](http://library.wolfram.com/infocenter/TechNotes/174)

---

T. Hahn, Introduction to Mathematica – p.21
Summary

- Mathematica makes it wonderfully easy, even for fairly unskilled users, to manipulate expressions.

- Most functions you will ever need are already built in. Many third-party packages are available at MathSource, http://library.wolfram.com/infocenter/MathSource.

- When using its capabilities (in particular list-oriented programming and pattern matching) right, Mathematica can be very efficient.

  Wrong: FullSimplify[veryLongExpression].

- Mathematica is a general-purpose system, i.e. convenient to use, but not ideal for everything. For example, in numerical functions, Mathematica usually selects the algorithm automatically, which may or may not be a good thing.
Books

- M. Trott, The Mathematica Guidebook
  - The Mathematica Guidebook for Programming
  - The Mathematica Guidebook for Graphics
  - The Mathematica Guidebook for Numerics
  - The Mathematica Guidebook for Symbolics
- R. Maeder, Programming in Mathematica