The Pure $\TeX_{\text{MACS}}$ Plugin

Albert Gräf

Email: `dr.graef@t-online.de`
Web: `http://pure-lang.googlecode.com/`

February 28, 2018

Table of contents

1. About Pure

Pure is a modern-style functional programming language based on term rewriting. It offers equational definitions with pattern matching, full symbolic rewriting capabilities, dynamic typing, eager and lazy evaluation, lexical closures, built-in list and matrix support and an easy-to-use C interface. The interpreter uses LLVM as a backend to JIT-compile Pure programs to fast native code.

Pure is by itself a very advanced language for symbolic computations. In addition, both Octave and Reduce can be run as embedded components in Pure, which creates a nicely integrated and powerful environment for scientific computing. But Pure gives you much more than that; it provides you with a full-featured functional programming environment with a fairly comprehensive collection of add-on modules for all major areas of computing, and the ability to interface to other 3rd party software quite easily if needed.

The integration with $\TeX_{\text{MACS}}$ adds another dimension by letting you write Pure programs in a style which looks just like mathematical definitions. And these formulas don't just sit there looking nice, they can be executed, too! The plugin supports all major features of the $\TeX_{\text{MACS}}$ interface, including Pure sessions and scripting, completion of Pure keywords and function names, accessing the Pure online help facility, as well as mathematical input and output (the latter is implemented using the Reduce `tmprint` package and thus requires Reduce). The examples in this document show off some of Pure's symbolic computing capabilities in $\TeX_{\text{MACS}}$, using Pure's onboard facilities as well as the Reduce interface which nicely integrates with Pure and $\TeX_{\text{MACS}}$.[1]

ACKNOWLEDGMENTS. Thanks are due to Kurt Pagani who provided much help, mission-critical Reduce/Lisp code and documentation for the Reduce and $\mathrm{T_EX_{MACS}}$ interfaces. Without his perseverance, insight and encouragment this plugin wouldn't exist.

2. Pure Sessions

You can insert a Pure session as usual with Insert | Session | Pure.

```
  __ \  |   |  __| _ \    Pure 0.56 (x86_64-unknown-linux-gnu)
  |   | |   | |    __/    Copyright (c) 2008-2012 by Albert Graef
  .__/ \__,_|_|  \___|    (Type 'help' for help, 'help copying'
 _|                       for license information.)

 Loaded prelude from /usr/lib/pure/prelude.pure.

> fact n = if n>0 then n*fact (n-1) else 1;
> map fact (0..10);
[1,1,2,6,24,120,720,5040,40320,362880,3628800]
> show fact
fact n = if n>0 then n*fact (n-1) else 1;
```

As with other $\mathrm{T_EX_{MACS}}$ session inserts, these snippets aren't just verbatim Pure code, they are *real interactions* with the Pure interpreter, so you can rerun the calculations or enter your own code. By these means you can use $\mathrm{T_EX_{MACS}}$ as a frontend for the Pure interpreter; please check the $\mathrm{T_EX_{MACS}}$ documentation, section "$\mathrm{T_EX_{MACS}}$ as an interface", for details. To make this work, you'll have to install the plugin first so that $\mathrm{T_EX_{MACS}}$ knows about it; instructions for that can be found in the Pure installation instructions. The distributed configuration actually defines various different types of Pure sessions, each with their own options for the Pure interpreter, and it's easy to add your own if needed.

Sessions can be formatted in different ways. Here we use the $\mathrm{T_EX_{MACS}}$ varsession style package for a somewhat fancier formatting. It's also possible to globally override formatting options such as the color of prompts, input and output fields, by defining the `pure-input` and `pure-output` macros accordingly; see the $\mathrm{T_EX_{MACS}}$ manual, section "Writing $\mathrm{T_EX_{MACS}}$ style files", for details. An example can be found in the accompanying `pure-session-styles.ts` file; install this in your `~/.TeXmacs/packages` directory if you want to give it a try.

Here's another session showing plain text formatting and subsessions.

```
  __ \  |   |  __| _ \    Pure 0.56 (x86_64-unknown-linux-gnu)
  |   | |   | |    __/    Copyright (c) 2008-2012 by Albert Graef
  .__/ \__,_|_|  \___|    (Type 'help' for help, 'help copying'
 _|                       for license information.)

 Loaded prelude from /usr/lib/pure/prelude.pure.

> fact n = if n>0 then n*fact (n-1) else 1;
```

⇑ This is a subsession.

1. Note that the examples in each section are to be executed in the given order, as some calculations rely on earlier definitions.

`> map fact (1..10); //` This is a *plain text* comment with math: $n! = 1 \times \cdots \times (n-1) \times n$.

```
[1,2,6,24,120,720,5040,40320,362880,3628800]
> show fact
fact n = if n>0 then n*fact (n-1) else 1;
>
> fact fact;
<stdin>, line 4: unhandled exception 'failed_cond' while evaluating 'fact
fact'
> fact 30L;
265252859812191058636308480000000L
> quit
```

↯ Dead

Another session type (**`pure-debug`**, this runs the interpreter in debugging mode):

```
 __ \  |   |  __| _ \    Pure 0.56 (x86_64-unknown-linux-gnu)
 |   | |   | |    __/    Copyright (c) 2008-2012 by Albert Graef
 .__/ \__,_|_|  \___|    (Type 'help' for help, 'help copying'
_|                       for license information.)

Loaded prelude from /usr/lib/pure/prelude.pure.

> fact n = if n>0 then n*fact (n-1) else 1;
> break fact
> fact 5;
** [1] fact: fact n = if n>0 then n*fact (n-1) else 1;
     n = 5
(Type 'h' for help.)
: h
Debugger commands:
a       auto: step through the entire program, run unattended
c [f]   continue until next breakpoint, or given function f
h       help: print this list
n       next step: step over reduction
p [n]   print rule stack (n = number of frames)
r       run: finish evaluation without debugger
s       single step: step into reduction
t, b    move to the top or bottom of the rule stack
u, d    move up or down one level in the rule stack
x       exit the interpreter (after confirmation)
.       reprint current rule
! cmd   execute interpreter command
? expr  evaluate expression
<cr>    single step (same as 's')
<eof>   step through program, run unattended (same as 'a')
: s
```

```
** [2] (>): x::int>y::int = x>y;
     x = 5; y = 0
```

```
: s
```

```
++ [2] (>): x::int>y::int = x>y;
     x = 5; y = 0
     --> 1
** [2] (-): x::int-y::int = x-y;
     x = 5; y = 1
```

```
: s
```

```
++ [2] (-): x::int-y::int = x-y;
     x = 5; y = 1
     --> 4
** [2] fact: fact n = if n>0 then n*fact (n-1) else 1;
     n = 4
```

```
: r
```

```
120
```

Note that TeXmacs always runs plugins in the current working directory in which it was started, which is also the directory from which the Pure interpreter will read its startup files. If you start TeXmacs from the GUI, this will most likely be your home directory. It often makes more sense to run the interpreter in the directory of the TeXmacs document hosting the Pure sessions, so that you can keep scripts and other files needed by your sessions in the same directory. Unfortunately, TeXmacs offers no help with this; there's simply no way to specify a working directory when running a session. Thus you'll either have to run TeXmacs in the right directory (by invoking it from the command line), or change the working directory manually inside the Pure session. To help with the latter, the TeXmacs-hosted interpreter offers a special `cdd` ("current document directory") command:

```
> pwd
```

```
/home/ag
```

```
> cdd
> pwd
```

```
/home/ag/.TeXmacs/plugins/pure/doc
```

3. Completion and Help

Completion of Pure keywords and functions is fully supported in TeXmacs. Just type `Tab` as usual and TeXmacs displays a list of possible completions in its status line. Pressing `Tab` again you can cycle through the completions and pick the one that you want. For instance, you can try this yourself on the following input line by placing the cursor behind the `f` and hitting the `Tab` key repeatedly:

```
> f
```

The Pure `help` command also works in TeXmacs. This pops up a new TeXmacs window with the help file in it. Search terms also work as usual; you might want to try the following to find out how the Pure help system works (this may take a while to load, so be patient):

```
> help online-help
```

Note that Pure's online help is in html format by default. While TeXmacs can load html files, it has to convert them to its own format first, which at least in the current version of TeXmacs is quite slow and the rendering isn't perfect. As a remedy, there's TeXmacs-formatted online documentation available on the Pure website, see the Pure installation instructions for details. If these files are installed then the Pure help can also be accessed using the corresponding options in the Pure plugin menu which automatically appears when the cursor is inside a Pure session. If you still prefer the html documentation, then it's also possible to use an external graphical html browser instead. Just set the `PURE_HELP` shell environment variable accordingly, the interpreter will then use that command to display the online help.[2]

4. Pure and Reduce

The following example shows how to run the Reduce computer algebra system in Pure (to make this work, you also need to have the `pure-reduce` module installed; this is available as an addon from the Pure website).

```
 __ \  |   |  __| _ \    Pure 0.56 (x86_64-unknown-linux-gnu)
 |   | |   | |    __/    Copyright (c) 2008-2012 by Albert Graef
 .__/ \__,_|_|  \___|    (Type 'help' for help, 'help copying'
_|                       for license information.)

Loaded prelude from /usr/lib/pure/prelude.pure.

> using reduce;

Reduce (Free CSL version), 03-Nov-12 ...

> simplify (df (sin (x^2)) x);

2*cos (x^2)*x
```

Up to now we've only been running Pure in verbatim a.k.a. program mode. But the Pure plugin also fully supports math input and output. These are enabled as follows:

- To use math *input*, you can toggle the input line between math and program (verbatim) mode using the `Ctrl+Shift+M` key combination. This isn't a standard TeXmacs keybinding, but is defined at the beginning of the `pure-init.scm` script for your convenience; you can edit the script to change this according to your preferences. Of course, you can also use the corresponding Focus | Input options | Mathematical input menu option or the equivalent toolbar item; these become visible when the cursor is located at the input line. Or you can make math input the default by putting the following Scheme command into your `my-init-texmacs.scm` file:

  ```
  (if (not (session-math-input?)) (toggle-session-math-input))
  ```

- To enable math *output*, you'll have to invoke the `math` function from the Pure `texmacs` module. This module is always loaded when running Pure inside TeXmacs. The `math` function uses the Reduce `tmprint` package, so the `pure-reduce` module *must* be installed to make this work.

  The `verbatim` function switches back to verbatim Pure output. Verbatim output is also used as a fallback in math mode for all Pure expressions which cannot be printed through the Reduce interface (typically because they aren't valid Reduce expressions).

2. Note that the `PURE_HELP` environment variable also works outside of TeXmacs, i.e., when the interpreter is run from Emacs or the shell. The `BROWSER` environment variable, however, does *not* change the way that the `help` command works in TeXmacs, so you can use this variable instead to specify a browser program for use outside of TeXmacs only.

> To make math output the default, you can also run a `pure-math` session (Insert | Session | Pure-math) which works like a regular Pure session but enables math output and loads the `reduce` module at startup.

At present this is still a bit experimental and work in progress, but it seems to work pretty well already, as shown below. (If you notice any bugs or missing features in math input and output, please submit a bug report on the Pure website.)

```
> math;
```

```
()
```

> $\mathrm{simplify}\,(\mathrm{df}\,(\sin(x^2))\,x);$

$2\cos(x^2)\,x$

> $\mathrm{simplify}\,(\mathrm{intg}\,(\cos(x+y)^2)\,x);$

$$\frac{\cos(x+y)\sin(x+y)+x}{2}$$

A more detailed account on math input and output can be found in the following section.

5. Mathematical Input and Output

For the purposes of this section, let's start up a regular Pure session and load the `math` and `reduce` modules.

```
 __ \  |   |  __| _ \    Pure 0.56 (x86_64-unknown-linux-gnu)
 |   | |   | |    __/    Copyright (c) 2008-2012 by Albert Graef
 .__/ \__,_|_|  \___|    (Type 'help' for help, 'help copying'
_|                       for license information.)

Loaded prelude from /usr/lib/pure/prelude.pure.
```

```
> using math, reduce;
Reduce (Free CSL version), 03-Nov-12 ...
> simplify (df (sin (x^2)) x);
2*cos (x^2)*x
```

To enter an expression like the one above as a mathematical formula, we must first switch the input line to *math input mode*. To do that, you can go search the toolbar for Input Options | Mathematical Input and check it, or type the key combination `Ctrl+Shift+M` defined by the Pure plugin. Another useful convenience is the `?` prefix operator (defined in `texmacs.pure`) which simplifies its expression argument, which is quoted automagically. Here's how the expression `? df (sin (x^2)) x)` looks like when typed in math input mode:

> $?\mathrm{df}\,(\sin(x^2))\,x;$

```
2*cos (x^2)*x
```

This is pretty much the same as:

> $\mathrm{simplify}\,('\mathrm{df}\,(\sin(x^2))\,x);$

```
2*cos (x^2)*x
```

But it's a lot easier to type, and the `?` operator uses `simplifyd` rather than just `simplify`, which also supports some customary notation for limits, integrals and differentials commonly used in TeXmacs. The `?:` operator does the same, but evaluates its argument; you want to use that if the expression includes some Pure functions which should be evaluated before submitting the result to Reduce. Note the difference:

> $\text{foo}\,x = x + 1;$

> $?\text{intg}(\text{foo}\,x)\,x;$

```
*** foo declared operator
intg (foo x) x
```

> $?\!:\text{intg}(\text{foo}\,x)\,x;$

```
(x^2+2*x)/2
```

Note that both `?` and `?:` are at the lowest possible precedence level in Pure so that their arguments don't have to be parenthesized (but this also means that you'll have to parenthesize the entire `?` or `?:` expression if you want to use it in the context of a larger expression).

Once we change to *math output mode*, simplifications of expressions are done automatically by the pretty-printer, so we can often do without explicitly using the `simplify`, `?` and `?:` operations. We'll do the rest of this session in this mode, so let's switch to it now. As explained in the previous section, this is done by invoking the `math` function from the `texmacs` module.[3]

> $\text{let math};$

> $\text{df}\,(\sin(x^2))\,x;$

$2\cos(x^2)\,x$

Note that the `df` term got simplified, but that's only in the display:

> $\text{verb ans};$

```
df (sin (x^2)) x
```

To actually obtain a simplified result which can be processed further in Pure land (passed to Pure functions or assigned to Pure variables), you still need to use `?`, `?:` etc.:

> $?\text{df}\,(\sin(x^2))\,x;$

$2\cos(x^2)\,x$

> $\text{verb ans};$

```
2*cos (x^2)*x
```

Note that the function `verb` from the `texmacs` module lets us peek at how the verbatim result looks like, without actually switching to verbatim mode. Similarly, the `mth` function temporarily switches the output mode to math mode when in verbatim mode.

There are some Pure expressions (such as strings, lambdas and pointers) which are always printed verbatim. However, the pretty-printer will try to print as much of the result in math mode as it can. For instance:

```
> (\x->x^2),"abc",df (sin (x^2)) x*y;
```

3. Note the `let` keyword before the call to `math`. In Pure this normally indicates a global variable definition, but if the left-hand side is omitted, the interpreter executes the right-hand side expression as usual and skips the printing of the result, similar to what the `$` terminator does in Reduce. We'll use this trick a lot in the following, to suppress uninteresting evaluation results.

`#<closure 0x7f1db00e3580>,"abc",`$2\cos(x^2)\,x\,y$

There are in fact two different variations of math output mode. Just calling `math` is the same as `algebraic` which corresponds to Reduce's *algebraic mode* and causes printed expressions to be simplified before they are printed. In contrast, `symbolic` employs Reduce's *symbolic mode* to pretty-print the raw expressions *without* simplifying them.[4] Likewise, there are functions `alg` (which is a synonym for `mth`) and `symb` to temporarily switch to the algebraic and symbolic math output modes, respectively. For instance:

> $\mathrm{df}\,(\sin(x^2))\,x;$ `// still in algebraic math mode`

$2\cos(x^2)\,x$

> $\mathrm{symb\,ans};$ `// the same in symbolic math mode`

$$\frac{\partial\sin(x^2)}{\partial x}$$

> $\mathrm{verb\,ans};$ `// the same as a verbatim Pure expression`

```
df (sin (x^2)) x
```

Note that there are some kinds of symbolic expressions (specifically, the logical expressions other than equality) which aren't by default valid in Reduce's algebraic mode, but work in symbolic mode:

> $x \geqslant y \wedge y > 0;$

x`>=`y`&&`y`>`0

> $\mathrm{symb\,ans};$

$x \geq y \wedge y > 0$

> $?x > 0;$

```
***** > invalid as algebraic operator

***** > invalid as algebraic operator
```

x`>`0

As a remedy, you can just declare the logical operations as Reduce operators; this won't actually convince Reduce to simplify them, but at least it gets rid of those annoying error messages and also makes the pretty-printer do a nicer job in algebraic mode:

> $\mathrm{declare\,operator}\,[(\neq),(<),(>),(\leqslant),(\geqslant),(\neg),(\wedge),(\vee)];$

```
()
```

> $x \geqslant y \wedge y > 0;$

$x \geq y \wedge y > 0$

> $?x \geqslant y \wedge y > 0;$

$x \geq y \wedge y > 0$

5.1. Basic expressions

Most mathematical expressions are mapped to corresponding Pure expressions in a sensible way.

[4]. Note that these commands don't actually change Reduce's internal mode of operation, they only affect the display mode in $\mathrm{T_EX_{MACS}}$.

We start out by declaring a few additional operators to be used below, so that they are known to Reduce. (This isn't strictly necessary, but silences the "`declared operator`" messages from Reduce.)

> $\text{let declare operator [above, below, binom]};$

> $(\backslash x \to 2\,x + \alpha)\,5;$ `// lambdas`

$\alpha + 10$

> $x_k, x^k, {}_k x, {}^k x, \underset{k}{x}, \overset{k}{x};$ `// sub- and superscripts`

$x\texttt{!}k, x^k, x\texttt{!}k, x^k, \text{below}(x,k), \text{above}(x,k)$

> $\frac{x}{y}, \frac{x}{y}, \dfrac{x}{y}, {}^x\!/_y, x\,/\,y;$ `// fractions`

$\dfrac{x}{y}, \dfrac{x}{y}, \dfrac{x}{y}, \dfrac{x}{y}, \dfrac{x}{y}$

> $189 \div 30;$ `// exact division p%q, yields Pure rationals`

$\dfrac{63}{10}$

> $\binom{n-1}{k-1} + \binom{n-1}{k};$ `// binomials`

$\text{binom}(n-1, k-1) + \text{binom}(n-1, k)$

> $\begin{pmatrix} a & b \\ b & -a \end{pmatrix}, \begin{matrix} a & b \\ b & -a \end{matrix}, \begin{array}{|c|c|} \hline a & b \\ \hline b & -a \\ \hline \end{array};$ `// matrices and tables (various formats)`

$\begin{pmatrix} a & b \\ b & -a \end{pmatrix}, \begin{pmatrix} a & b \\ b & -a \end{pmatrix}, \begin{pmatrix} a & b \\ b & -a \end{pmatrix}$

> $\begin{vmatrix} a & b \\ b & -a \end{vmatrix};$ `// determinants`

$-(a^2 + b^2)$

> $\begin{pmatrix} a & b & c \end{pmatrix}, \begin{pmatrix} a \\ b \\ c \end{pmatrix};$ `// vectors`

$\begin{pmatrix} a & b & c \end{pmatrix}, \begin{pmatrix} a \\ b \\ c \end{pmatrix}$

> $\exp(x)\,\alpha + \beta;$ `// arithmetic`

$e^x\,\alpha + \beta$

> $\sqrt{x^2 + y^2}, \sqrt[3]{x};$ `// roots`

$\sqrt{x^2 + y^2}, \sqrt[3]{x}$

> $\neg A \wedge (B \vee C);$ `// logic`

$\neg(A) \wedge (B \vee C)$

> $x > y, x < y, x \geqslant y, x \leqslant y, x == y, x \neq y;$ `// comparisons`

$x > y, x < y, x \geq y, x \leq y, x == y, x \neq y$

> $'(x \equiv y, x \not\equiv y);$ `// syntactic equality (=== / ~== in Pure)`

$x\texttt{===}y, x\texttt{~==}y$

> eval ans;

$0,1$

> $[1,2,3]$; `// lists`

$[1,\ 2,\ 3]$

> $1\ldots10; 1{:}3\ldots11$; `// arithmetic sequences`

$[1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9,\ 10]$
$[1,\ 3,\ 5,\ 7,\ 9,\ 11]$

> $[a+1 \mid a=1\ldots10; a \bmod 2]$; `// comprehensions`

$[2,\ 4,\ 6,\ 8,\ 10]$

> $(a,b+1); [a,b+1]; \langle a,b+1\rangle; [\![a,b+1]\!]; \lfloor n/2\rfloor; \lceil n/2\rceil; |x|; \|x\|$; `// various brackets`

$a,b+1$
$[a,\ b+1]$
$a,b+1$
$[a,\ b+1]$
$\mathrm{floor}\left(\frac{n}{2}\right)$
$\mathrm{ceiling}\left(\frac{n}{2}\right)$
$\mathrm{abs}(x)$
$\mathrm{norm}(x)$

> let declare operator [hat, tilde, bar, vect, check, breve, dot, ddot, acute, grave];

> $\hat{x}; \tilde{x}; \bar{x}; \vec{x}; \check{x}; \breve{x}; \dot{x}; \ddot{x}; \acute{x}; \grave{x};' x;'' x; \not{x}; \sim x$; `// accents, primes, etc.`

$\mathrm{hat}(x)$
$\mathrm{tilde}(x)$
$\mathrm{bar}(x)$
$\mathrm{vect}(x)$
$\mathrm{check}(x)$
$\mathrm{breve}(x)$
$\mathrm{dot}(x)$
$\mathrm{ddot}(x)$
$\mathrm{acute}(x)$
$\mathrm{grave}(x)$
x
$'x$
$\neg(x)$
$\neg(x)$

> $x\oplus y; x\ominus y; x\otimes y; x\oslash y; x\pm y; x\mp y; x\div y; x\cap y; x\cup y; x\uplus y$; `// infix operators`

x `oplus` y
x `ominus` y
x `otimes` y
x `oslash` y
x `pm` y
x `mp` y
$x\%y$
x `cap` y
x `cup` y
x `uplus` y

> $\alpha; \Gamma; \mathbf{Z}; \boldsymbol{z}; \mathbf{0}; \mathcal{C}; \mathfrak{F}; \mathfrak{u}, \mathfrak{v}, \mathfrak{w}; \mathbb{Q}$; `// Greek symbols, special glyphs`

α
Γ
Z
z
0
C
F
u,v,w
QQ

> **foo**, *bar*, *baz*, `gnu`, **`gna`**, gnats; `// special markup`

foo,bar,baz,gnu,gna,gnats

> $\scriptscriptstyle A,B$, $\scriptstyle A,B$, $\scriptstyle A,B$, A,B, A,B, $\large A,B$, $\Large A,B$, $\LARGE A,B$, $\huge A,B$, $\Huge A,B$; `// various sizes`

$A,B,A,B,A,B,A,B,A,B,A,B,A,B,A,B,A,B,A,B$

More examples using list and matrix comprehensions (these are all plain Pure evaluations, Reduce is only used for the pretty-printing here):[5]

> $\|X::\mathrm{matrix}\| = \sqrt{\mathrm{sum}\,[x^2|\,x=X]}$;

> $\|(\begin{smallmatrix}1 & 2 & 3 & 4\end{smallmatrix})\|$;

5.47722557505

> $[2\,x+1|x=1\ldots5]$;

[3, 5, 7, 9, 11]

> $\mathrm{eye}\,n = (\,i==j\,|\,i=1\ldots n;\, j=1\ldots n\,)$;

> $\mathrm{eye}\,3$;

$$\begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{pmatrix}$$

Here's an example of an infinite list, a prime sieve:

> **let** $P=\mathrm{sieve}\,(2\ldots\infty)$ **with** $\mathrm{sieve}(p\colon qs)=p\colon\mathrm{sieve}[q\,|\,q=qs;\,q\,\mathsf{mod}\,p]\,\&$ **end**;

> $P!!(0\ldots20)$;

[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71
, 73]

> $\mathrm{list}\,(\mathrm{take}\,20\,(\mathrm{drop}\,100\,P))$;

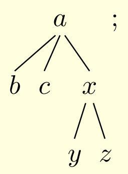
[547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631,
641, 643, 647, 653, 659]

5.2. Trees

TeXmacs has a nice and convenient notation for labelled trees. These are handled gracefully in Pure as well:

> let declare operator tree;

5. Note that in order to get the single vertical bar | in math mode, you'll have to type either `Shift+F5 |` or `Alt+M |` (the latter gives you a “big” | symbol which automatically expands with the brackets surrounding it).

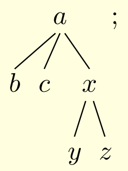
> 

;

tree$(a, b, c, \text{tree}(x, y, z))$

Because `tree` is variadic (a tree may have any number of subtrees), it's denoted as an uncurried function in Pure. The first argument of `tree` is always the label of the root node, the remaining arguments are the subtrees and/or leaves beneath the root node. The simplest way to translate this into a nested Pure list for easier processing is to just define `tree` as the standard Pure function `list`:

> tree = list;

> 

;

$[a, b, c, [x, y, z]]$

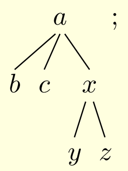
This corresponds to a preorder traversal of the tree. If you prefer to have the labels of the branches as separate arguments then you might use a definition like the following instead:

> clear tree

> declare operator branch;

```
()
```

> tree $(x, ys :: \text{tuple}) = \text{branch}\ x\ (\text{list}\ ys)$; tree $(x, y) = \text{branch}\ x\ [y]$;

> 

;

branch$(a, [b, c, \text{branch}(x, [y, z])])$

Note that the way we defined `tree`, `branch` is in fact a curried constructor; the uncurried notation is an artifact of the pretty-printing here. As a verbatim Pure term the result looks like this:

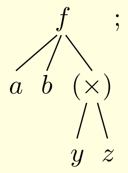
> verb ans;

```
branch a [b,c,branch x [y,z]]
```

If the trees denote valid Pure terms then you might wish to get rid of all the extra baggage and just translate them to plain Pure applications. Here's a third variation of `tree` which does this:

> clear tree

> tree x = foldl1 (\$) (list x);

> 

;

```
*** f declared operator
```

$f\,(a, b, y\,z)$

> verb ans;

```
f a b (y*z)
```

With this definition of `tree` you can now enter arbitrary Pure expressions as trees if you like:

```
>    (^)  ;
     / \
   (+)  2
   / \
   a  b
```

$a^2 + 2\,a\,b + b^2$

> verb ans;

```
(a+b)^2
```

5.3. Lists

List and vector/matrix data can be exchanged between Pure and Reduce in a seamless fashion. This makes it easy to inspect and manipulate compound results returned by Reduce, such as lists of partial fractions:

> let $p = 2/((x+1)^2\,(x+2))$; p;

$$\frac{2}{x^3 + 4\,x^2 + 5\,x + 2}$$

> let off exp; p; `// expansion switched off`

$$\frac{2}{(x+2)\,(x+1)^2}$$

> let pfs = ?: pf p x; pfs;

$$\left[\frac{2}{x+2}, \frac{-2}{x+1}, \frac{2}{(x+1)^2}\right]$$

> map ($\backslash y \rightarrow$ df y x) pfs;

$$\left[\frac{-2}{(x+2)^2}, \frac{2}{(x+1)^2}, \frac{-4}{(x+1)^3}\right]$$

> let on exp; ans; `// expansion switched back on again`

$$\left[\frac{-2}{x^2 + 4\,x + 4}, \frac{2}{x^2 + 2\,x + 1}, \frac{-4}{x^3 + 3\,x^2 + 3\,x + 1}\right]$$

Let's consider another typical example, equation solving:

> let eqn = $\log(\sin(x+3))^5 == 8$; eqn;

$\log\,(\sin\,(x+3))^5 == 8$

> let solns = ?: solve eqn x;

> $\#\mathrm{solns}; \mathrm{take}\, 2\, \mathrm{solns};$

10

$$\left[x = 2\ \mathrm{arbint}(5)\ \pi + \arcsin\left(\frac{\exp\left(2^{3/5}\cos\left(\frac{2\,\pi}{5}\right)\right)}{\exp\left(2^{3/5}\sin\left(\frac{2\,\pi}{5}\right)i\right)}\right) - 3,\ x = 2\ \mathrm{arbint}(5)\ \pi - \arcsin\left(\frac{\exp\left(2^{3/5}\cos\left(\frac{2\,\pi}{5}\right)\right)}{\exp\left(2^{3/5}\sin\left(\frac{2\,\pi}{5}\right)i\right)}\right) + \pi - 3\right]$$

> $\mathrm{let}\, \mathrm{solns} = ?{:}\, \mathrm{reduce\_with}\, [\mathrm{arbint} \Rightarrow \mathrm{cst}\, 0]\ \mathrm{solns}; \mathrm{take}\, 2\, \mathrm{solns};$

$$\left[x = \arcsin\left(\frac{\exp\left(2^{3/5}\cos\left(\frac{2\,\pi}{5}\right)\right)}{\exp\left(2^{3/5}\sin\left(\frac{2\,\pi}{5}\right)i\right)}\right) - 3,\ x = -\arcsin\left(\frac{\exp\left(2^{3/5}\cos\left(\frac{2\,\pi}{5}\right)\right)}{\exp\left(2^{3/5}\sin\left(\frac{2\,\pi}{5}\right)i\right)}\right) + \pi - 3\right]$$

> $\mathrm{check}\ (u == v)\, (x == y) = \mathrm{eval}(?{:}\, \mathrm{reduce\_with}\, [x \Rightarrow y]\, |u - v|);$

> $\Delta\, s = \mathrm{check}\, \mathrm{eqn}\ s; \mathrm{let}\, \varepsilon = 10^{-8};$

> $[y\, |\, x == y = \mathrm{solns}; \Delta(x == y) == 0];$ `// exact zeros`

$$\left[\arcsin\left(e^{2^{3/5}}\right) - 3,\ -\arcsin\left(e^{2^{3/5}}\right) + \pi - 3\right]$$

> $[y\, |\, s@(x == y) = \mathrm{solns}; \Delta\, s \neq 0 \wedge \Delta\, s \leqslant \varepsilon];$ `// inexact zeros`

$$\left[\arcsin\left(\exp\left(2^{3/5}\cos\left(\frac{2\,\pi}{5}\right) + 2^{3/5}\sin\left(\frac{2\,\pi}{5}\right)i\right)\right) - 3\right]$$

> $[y \Rightarrow \Delta\, s\, |\, s@(x == y) = \mathrm{solns}; \Delta\, s > \varepsilon];$ `// what's up with these??`

$$\left[\arcsin\left(\frac{\exp\left(2^{3/5}\cos\left(\frac{2\,\pi}{5}\right)\right)}{\exp\left(\left(2^{3/5}\sin\left(\frac{2\,\pi}{5}\right)\right)i\right)}\right) - 3 \Rightarrow 7.8764,\ \left(-\arcsin\left(\frac{\exp\left(2^{3/5}\cos\left(\frac{2\,\pi}{5}\right)\right)}{\exp\left(\left(2^{3/5}\sin\left(\frac{2\,\pi}{5}\right)\right)i\right)}\right) + \pi\right) - 3 \Rightarrow 7.8764,\right.$$
$$\arcsin\left(\frac{1}{\exp\left(2^{3/5}\cos\left(\frac{\pi}{5}\right) + \left(2^{3/5}\sin\left(\frac{\pi}{5}\right)\right)i\right)}\right) - 3 \Rightarrow 7.8764,\ \left(-\arcsin\left(\frac{1}{\exp\left(2^{3/5}\cos\left(\frac{\pi}{5}\right) + \left(2^{3/5}\sin\left(\frac{\pi}{5}\right)\right)i\right)}\right) + \pi\right) - 3 \Rightarrow 7.8764,$$
$$\arcsin\left(\frac{\exp\left(\left(2^{3/5}\sin\left(\frac{\pi}{5}\right)\right)i\right)}{e^{2^{3/5}\cos\left(\frac{\pi}{5}\right)}}\right) - 3 \Rightarrow 7.8764,\ \left(-\arcsin\left(\frac{\exp\left(\left(2^{3/5}\sin\left(\frac{\pi}{5}\right)\right)i\right)}{e^{2^{3/5}\cos\left(\frac{\pi}{5}\right)}}\right) + \pi\right) - 3 \Rightarrow 7.8764,$$
$$\left.\left(-\arcsin\left(\exp\left(2^{3/5}\cos\left(\frac{2\,\pi}{5}\right) + \left(2^{3/5}\sin\left(\frac{2\,\pi}{5}\right)\right)i\right)\right) + \pi\right) - 3 \Rightarrow 7.8764\right]$$

5.4. Big operators (integrals, limits, sums, etc.)

Big operators (Insert | Symbol | Big operator in math mode) are mapped to corresponding Pure expressions, generally using a Reduce-compatible form:

> $\int \sin x\, \mathrm{d}x;$

$-\cos(x)$

```
> reduce::load "defint"; // we need this for the definite integrals
```

0

> $\int_a^b x^2 \mathrm{d}x; \int_a^b x^2 \mathrm{d}x;$

$$\frac{-a^3+b^3}{3}$$
$$\frac{-a^3+b^3}{3}$$

> $\lim_{x\to 0}(1/x), \lim_{x\to\infty}(x\sin(1/x)), \lim_{x\to\infty}\frac{1}{x};$

$\infty,1,0$

> $\int \sin x\,\mathrm{d}x, \int_a^b x^2\mathrm{d}x, \int_a^b x^2\mathrm{d}x;$

$-\cos(x), \frac{-a^3+b^3}{3}, \frac{-a^3+b^3}{3}$

> $\sum_{k=1}^{n}(2\,k-1), \prod_{k=1}^{n}(2\,k-1);$

$n^2, \frac{2\,\gamma(2\,n)}{2^n\,\gamma(n)}$

> $\sum_{k=0}^{n-1}(a+k\,r), \prod_{k=1}^{n}\frac{k}{k+2};$

$\frac{n\,(2\,a+n\,r-r)}{2}, \frac{2}{n^2+3\,n+2}$

Sums and products with known (i.e., non-symbolic) bounds are translated to the appropriate aggregates of Pure list comprehensions so that they can be computed directly in Pure:

> $\sum_{k=1}^{5}(2\,k-1), \prod_{k=1}^{5}(2\,k-1);$

25,945

This is always the case if the entire generator clause is given as a subscript and the superscript is absent:

> $\sum_{k=1...5}(2\,k-1), \prod_{k=1...5}(2\,k-1);$

25,945

Note that the above is equivalent to the following verbatim Pure code:

```
sum [(2*k-1)|k=1..5], prod [(2*k-1)|k=1..5];
```

The same holds for a number of other big operators, such as the big wedge and vee, which have no counterpart in Reduce. These aren't predefined in Pure either, but we can implement some useful Pure operations with them, for instance:

> $\mathrm{bigwedge} = \mathrm{foldl}\,(\wedge)\,\mathrm{true}; \mathrm{bigvee} = \mathrm{foldl}\,(\vee)\,\mathrm{false};$

> $\bigwedge_{k=1}^{n}(x_{k-1}\geqslant 0), \bigvee_{k=1}^{n}(x_{k-1}<0)$ **when** $x=-3...3; n=\#x$ **end**;

0,1

5.5. Differentials and integrals

Both differentials and integrals can be specified using customary mathematical notation which gets translated to invocations of the Reduce `df` and `int` operators (the latter is actually named `intg` in Pure, to avoid a name clash with the built-in Pure function `int`). They are constructed according to the following syntactical rules:

- The differential operator is written as $d\ x$, which may be denoted either as a function application (with a space between d and x) or as a product with an (invisible) multiplication sign `*` between d and x. Instead of d, you can also use the upright d symbol (`\mathd` or `d` Tab Tab in math input mode), in which case no delimiter between d and x is needed. In the case of differentials you may also use the partial symbol ∂ (`\partial` or `d` Tab Tab Tab) instead.

- Integrals take the form $\int f\, \mathrm{d}\, x$ (`\big int` or `Shift+F5 ⇧I`) with the appropriate delimiters between f, d and x (either invisible multiplication signs or spaces; again these may be omitted if the special upright d symbol is used).

- Differentials are written as a quotient $\mathrm{d} f / \mathrm{d}x$, using either the `/` operator or an explicit fraction (`\frac` or `Alt+F`).[6] Higher-order differentials may be denoted either as an application $\mathrm{d}\, k\, f / \mathrm{d}\, x\, k$ or with a superscript $\mathrm{d}^k f / \mathrm{d}x^k$. In either case, multiple differentiation variables can be given as a product of the corresponding differential terms in the denominator, such as $\partial^2 f / (\partial x\, \partial y)$. Moreover, the notation $\mathrm{d}/\mathrm{d}x\ f$ (with a multiplication sign between $\mathrm{d}/\mathrm{d}x$ and f) is provided as an alternative to $\mathrm{d} f / \mathrm{d}x$.

- Note that if f or x are compound expressions, you may have to put them in parentheses according to the rules of Pure syntax. (If multiplication signs are used as delimiters and both f and x are either symbols or function applications, this is usually unnecessary since these bind stronger than both multiplication and exponents.)

Of course, in either case you may also just write the corresponding Pure/Reduce call, which is often easier to type, but doesn't nearly look as nice and mathematical. It is also instructive to take a look at how Reduce itself renders calls to **`intg`** and **`df`**; you can always copy such output to the input line again and it should just work. Here are some examples.

> $\text{let declare depend}\,[f, x];$

> $\text{df}\ f\, x\, 2, \text{intg}\ f\, x;$

$$\frac{\partial^2 f}{\partial x^2}, \int f\ d\ x$$

> $\partial^2 f / \partial x^2;$

$$\frac{\partial^2 f}{\partial x^2}$$

> $\int 2\, f(x) \mathrm{d}x;$

$$2 \int f(x)\ d\ x$$

> $\int 2\, f(x) \mathrm{d}\,(\cos x);$

$2 \cos(x)\ f(x)$

> $\mathrm{d}^2(x^3) / \mathrm{d}x^2;$

$6\, x$

> $\frac{\partial^9 (x^2\, y^3\, z^4)}{\partial x^2\, \partial y^3\, \partial z^4};$

288

> $\mathrm{d}/\mathrm{d}x\, (x + y)^5;$

$5\,(x^4 + 4\, x^3\, y + 6\, x^2\, y^2 + 4\, x\, y^3 + y^4)$

> $\mathrm{d}/\mathrm{d}x\, (\sin(x) \cos(x));$

$\cos(x)^2 - \sin(x)^2$

6. Note that in the former case the denominator $\mathrm{d}\, x$ has to be parenthesized if it is written in multiplicative form. This is because the multiplication operators including `*` and `/` are left-associative in Pure, so **`d*f/d*x`** will be parsed as **`((d*f)/d)*x`** rather than **`(d*f)/(d*x)`**. This pitfall can be avoided by just using an explicit fraction instead.

> $\int \sin(2\,x)\mathrm{d}x;$

$$\frac{-\cos(2\,x)}{2}$$

> $\int x^2\,(a\,x+b)^n\mathrm{d}x\;\mathbf{when}\;b=0\;\mathbf{end};$

$$\frac{x^n\,a^n\,x^3}{n+3}$$

It is often convenient to abbreviate parametric Reduce expressions using Pure functions. We'll learn about different ways to enter Pure functions in math mode below, but for the moment it suffices to know that the simplest form is just $f\;x_1\cdots x_n = \mathit{rhs}$ where f is the name of the function, x_1, ..., x_n are the parameters and *rhs* is the right-hand side (the body) of the definition. Note the missing parentheses around the parameters. Pure uses the curried notation for function applications where the parameters simply follow the function, similar to shell command syntax. For compatibility with Reduce, function calls in *Reduce* expressions can also be specified in the usual uncurried form $f(x_1, ..., x_n)$, but Pure definitions and expressions generally use the curried form. For instance:

> $I\;a\;b\;n = ?\int x^2\,(a\,x+b)^n\mathrm{d}x;$

Note that we employed the `?` operator here so that the `I` function produces a simplified result rather than just a literal integral which then needs to be computed when pretty-printed.

> $I a b n;$

$$\frac{(a\,x+b)^n\,(a^3\,n^2\,x^3+3\,a^3\,n\,x^3+2\,a^3\,x^3+a^2\,b\,n^2\,x^2+a^2\,b\,n\,x^2-2\,a\,b^2\,n\,x+2\,b^3)}{a^3\,(n^3+6\,n^2+11\,n+6)}$$

> $I a b 0;$

$$\frac{x^3}{3}$$

> $I 0 b n;$

$$\frac{b^n\,x^3}{3}$$

> $I a 0 k;$

$$\frac{x^k\,a^k\,x^3}{k+3}$$

5.6. Programming

We've already seen various simple kinds of Pure programs (i.e., function definitions) throughout this section. One important thing to note here is that all supported math elements not only work in expressions to be evaluated, but also when defining functions, on *both* sides of the definition. We can make good use of this to make Pure code look like real mathematical formulas. For instance, let's define a prettier notation for the list slicing operator (`!!` in Pure). We'd actually like to write an ordinary index in math mode, like this:

> $(\mathit{xs})_{1\ldots n};$

xs`!(1..`n`)`

As you can see, this kind of expression isn't readily defined in Pure, so we can do it ourselves:

> $(\mathit{xs}::\mathrm{list})_{\mathit{ys}::\mathrm{list}} = \mathit{xs}!!\mathit{ys};$

That's it. Now we can write:

> $(\text{"}a\text{"}...\text{"}z\text{"})_{10...16}$;

```
["k","l","m","n","o","p","q"]
```

> **let** $P = \text{sieve}\,(2...\infty)$ **with** $\text{sieve}(p{:}\,qs) = p{:}\,\text{sieve}[q\,|\,q = qs;\, q \;\mathsf{mod}\; p]\,\&$ **end**;

> $P_{99...117}$;

[541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619,
631, 641, 643, 647]

Let's consider another example, the binomials:

> $\binom{n}{k}$;

binom(n, k)

This function isn't predefined in Pure either, so let's do that now. To get nicely aligned equations, we'll use an equation array this time. This is available as `\eqnarray` in math mode; similarly, the binomials can be entered with `\binom` in math mode:

>
$$\begin{aligned} \binom{n::\text{int}}{k::\text{int}} &= \binom{n-1}{k-1} + \binom{n-1}{k} \;\textbf{if}\; n > k \wedge k > 0; \\ &= 1 \;\textbf{otherwise}; \end{aligned}$$

```
> show binom
```

```
binom n::int k::int = binom (n-1) (k-1)+binom (n-1) k if n>k&&k>0;
binom n::int k::int = 1;
```

Let's calculate the first five rows of the Pascal triangle to see that it works:

> $\left[\left[\binom{n}{k}\middle| k = 0...n\right]\middle| n = 0...5\right]$;

[[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1], [1, 5, 10, 10, 5, 1]]

Of course, the text book formula we used above isn't the best way to implement binomials. The following algorithm employing factorials is much faster; it also uses bigints to prevent wrapover.

> clear binom

> $\binom{n::\text{int}}{k::\text{int}} = \left(\prod_{i=k+1\text{L}}^{n} i\right) \text{div} \left(\prod_{i=1\text{L}}^{n-k} i\right)$;

> $\binom{30}{14}$;

145422675

As we already saw in the prime sieve example, Pure can deal with "lazy" lists (called *streams* in functional programming parlance) just fine. So let's be bold and just define the infinite stream of *all* rows of the Pascal triangle. This is easily done with a nested list comprehension in Pure:

> $\text{binomials} = \left[\left[\binom{n}{k}\middle| k = 0...n\right]\middle| n = 0...\infty\right]$;

> binomials; $\text{binomials}_{0...5}$; binomials_{30};

[1]:`#<thunk 0x7ff6b44e0ef0>`
[[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1], [1, 5, 10, 10, 5, 1]]
[1, 30, 435, 4060, 27405, 142506, 593775, 2035800, 5852925, 14307150,
30045015, 54627300, 86493225, 119759850, 145422675, 155117520, 145422675,
119759850, 86493225, 54627300, 30045015, 14307150, 5852925, 2035800, 593775
, 142506, 27405, 4060, 435, 30, 1]

Sometimes we may want to align subterms in expressions or definitions, in order to improve the readability of a formula. TeXmacs provides the stack construct for this purpose, which arranges stuff in rows and columns pretty much like a matrix. The Pure converter simply traverses this construct in row-major order. Thus the $\begin{matrix} x & y \\ z & t \end{matrix}$ stack will expand to just `x y z t` in Pure. This can be used, in particular, to align the parts of an equation or comprehension. For instance:

> $\left[\binom{n}{k} \middle| \begin{matrix} n & = & 0\ldots 5; \\ k & = & 0\ldots n \end{matrix}\right]$;

[1, 1, 1, 1, 2, 1, 1, 3, 3, 1, 1, 4, 6, 4, 1, 1, 5, 10, 10, 5, 1]

Note that the above produces exactly the same result as the following linear notation:

> $\left[\binom{n}{k} \middle| n = 0\ldots 5; k = 0\ldots n\right]$;

[1, 1, 1, 1, 2, 1, 1, 3, 3, 1, 1, 4, 6, 4, 1, 1, 5, 10, 10, 5, 1]

Moreover, the TeXmacs choice construct can be used to write Pure function definitions involving guards in a compact and pretty way. For instance, here's another definition of the factorial, this time entered in math mode:[7]

> $\mathrm{fact}(n) = \begin{cases} 1 & \textbf{if } n \leqslant 0 \\ n \times \mathrm{fact}(n-1) & \textbf{otherwise} \end{cases}$;

> $\mathrm{map}\,\mathrm{fact}\,(0\ldots 12)$;

```
[1,1,2,6,24,120,720,5040,40320,362880,3628800,39916800,479001600]
```

The above definition is in fact equivalent to the following verbatim Pure code (which isn't all that unreadable either, as Pure's function definition syntax already mimics mathematical notation very closely):

```
fact(n) = 1 if n<=0; = n*fact(n-1) otherwise;
```

Of course, the same construct can also be used to define local functions:

> $\mathrm{map}\,\mathrm{fact}\,(0\ldots 12)$ **with** $\mathrm{fact}(n) = \begin{cases} 1 & \textbf{if } n \leqslant 0 \\ n \times \mathrm{fact}(n-1) & \textbf{otherwise} \end{cases}$ **end**;

```
[1,1,2,6,24,120,720,5040,40320,362880,3628800,39916800,479001600]
```

Last but not least, the choice construct can also be used with Pure's pattern-matching `case` expressions. Note that the closing `end` of the `case` expression is omitted, the choice construct supplies it automatically.

> $|xs :: \mathrm{list}| = \textbf{case}\ xs\ \textbf{of} \begin{cases} [] & = & 0 \\ x{:}\, xs & = & 1 + |xs| & \textbf{otherwise} \end{cases}$;

> $|1{:}\, 3\ldots 100|$;

50

This defines $|xs|$ (a.k.a. `abs xs` in Pure notation) to compute the size of a list xs (similar to what Pure's `#` operator does). For instance, let's count the number of primes up to 5000 (this may take a little while; alas, our definition of the prime sieve isn't very efficient!):

> **let** $P = \mathrm{sieve}\,(2\ldots\infty)$ **with** $\mathrm{sieve}(p{:}\, qs) = p{:}\,\mathrm{sieve}[q\,|\,q = qs;\, q \bmod p]\,\&$ **end**;

7. Note that the **if** keyword is mandatory here, as it is required by the Pure syntax (as are the semicolons). The **otherwise** keyword is just syntactic sugar, however, although it often improves readability.

> $|\mathrm{takewhile}\,(\leqslant 5000)\,P|$;

669

5.7. Caveats

TeXmacs doesn't know anything about Pure syntax; as far as it is concerned, Pure's functions, operators and keywords are just text. So there are situations in which you may have to help the converter along by adding parentheses to disambiguate the parsing. This is true, in particular, for big operators (integrals, sums, etc., especially in conjunction with Pure `with` and `when` clauses) and differentials. Even an invisible bracket (shortcut: `(`Space) will be good enough. For instance:

```
> let math;
```

```
Reduce (Free CSL version), 03-Nov-12 ...
```

> $\mathrm{d}x^2/\mathrm{d}x$;

```
*** d declared operator
```

$d(x)$

Note the missing bracket around x^2. As `verb` reveals, this yields `(d x)^2` instead of `d (x^2)` (which is what we want):

> verb ans;

```
d x^2/d x
```

The remedy is to just parenthesize the x^2 term:

> $\mathrm{d}(x^2)/\mathrm{d}x$; // use parentheses around x^2 to disambiguate

$2\,x$

> $\mathrm{d}x^2/\mathrm{d}x$; // even an invisible bracket around x^2 does the trick

$2\,x$

Here's another snippet which produces a strange error:

> $\sum_{k=1\ldots n}(2\,k-1), \prod_{k=1\ldots n}(2\,k-1)$ **when** $n\!=\!5$ **end**;

```
<stdin>, line 6: syntax error, unexpected '|', expecting end
```

The tuple (comma operator) binds stronger than the `when` clause, so this looks like valid Pure syntax. What went wrong? Unfortunately, we can't use `verb` to reveal how Pure parsed the expression here, because it couldn't! But there's a neat trick to show us what exactly TeXmacs fed into the Pure interpreter: just copy and paste the entire expression to another input line operated in verbatim input mode. We get:

```
> sum [(2*k-1)|k=1..n],prod [(2*k-1) when n |k=1..n]= 5 end;
```

Note that for TeXmacs the "$(2\,k-1)$ **when** n" part looks like any other ordinary term belonging under the product on the right, which is followed by an equals sign and another term "5 **end**". This makes perfect sense for TeXmacs, but it's not valid Pure syntax. This wouldn't normally be a problem (Pure would be able to reparse the expression correctly anyway), if it wasn't for the $\prod$ operator which translates to a Pure list comprehension. So the "$(2\,k-1)$ **when** n" part ended up in this list comprehension where it doesn't belong, hence the somewhat surprising syntax error.

Placing brackets around either the entire tuple or just the product on the right correctly resolves the ambiguity. In this case, we might as well use visible parentheses, since they make the expression easier to parse for human readers, too:

> $(\sum_{k=1\ldots n}(2\,k-1), \prod_{k=1\ldots n}(2\,k-1))$ **when** $n{=}5$ **end**;

```
25,945
```

We can copy/paste the modified expression to a verbatim input line again, to confirm that it was converted correctly this time:

```
> (sum [(2*k-1)|k=1..n],prod [(2*k-1)|k=1..n]) when n = 5 end;
```

6. Pure and Octave

It's also possible to call Octave from Pure in order to do numeric calculations, and use the math output mode to handle the pretty-printing via Reduce. To do this, you also need to have the `pure-octave` package installed. For instance:

```
 __ \  |   |  __| _ \    Pure 0.56 (x86_64-unknown-linux-gnu)
 |   | |   | |    __/    Copyright (c) 2008-2012 by Albert Graef
 .__/ \__,_|_|  \___|    (Type 'help' for help, 'help copying'
_|                       for license information.)

Loaded prelude from /usr/lib/pure/prelude.pure.
```

```
> using octave; let math;
```

```
Reduce (Free CSL version), 03-Nov-12 ...
```

> $$\begin{aligned}\text{eig}\,x &= \text{octave\_call}\,\text{"eig"}\,1\,(\text{dmatrix}\,x);\\ \text{eig2}\,x &= \text{octave\_call}\,\text{"eig"}\,2\,(\text{dmatrix}\,x);\end{aligned}$$

> eig $\begin{pmatrix}1&2\\3&4\end{pmatrix}$; // eigenvalues only

$$\begin{pmatrix}-0.372281323269\\5.37228132327\end{pmatrix}$$

> eig2 $\begin{pmatrix}1&2\\3&4\end{pmatrix}$; // eigenvectors and diagonalized matrix

$$\begin{pmatrix}-0.824564840132 & -0.415973557919\\ 0.565767464969 & -0.909376709132\end{pmatrix}, \begin{pmatrix}-0.372281323269 & 0\\ 0 & 5.37228132327\end{pmatrix}$$

It goes without saying that this is pretty useful if a problem calls for a mix of symbolic and numeric algorithms. It's also possible to call back into Pure from Octave, and thereby into Reduce. To do this, you need to wrap up the computation as a Pure function which in turn uses `?` or `?:` to invoke Reduce. Just to illustrate how this works, here's a (somewhat contrived) example where we call the Reduce transpose function `tp` from Octave:

> tp x :: matrix = ?tp(x);

```
> octave_call "pure_call" 1 ("tp",{1,2;3,4});
```

$$\begin{pmatrix}1&3\\2&4\end{pmatrix}$$

```
> let octave_eval "pure_call('tp',[1 2;3 4])"; // Octave output!
ans =

   1   3
   2   4
```

In a similar fashion you might, e.g., have Octave call Pure to solve an equation which Octave itself can't handle.

With the `gnuplot.pure` module included in the latest version of the `pure-octave` package, you can also make good use of Octave's Gnuplot interface. There's a convenience function named `psplot` in the `texmacs` module which grabs an Octave plot in PostScript format and pipes it into $\mathrm{T_EX_{MACS}}$ so that it gets inserted directly into the session output. (The `psplot` function is in fact just a little wrapper around the Octave `print` function and the `ps` function from the `texmacs` module described in the following section.)

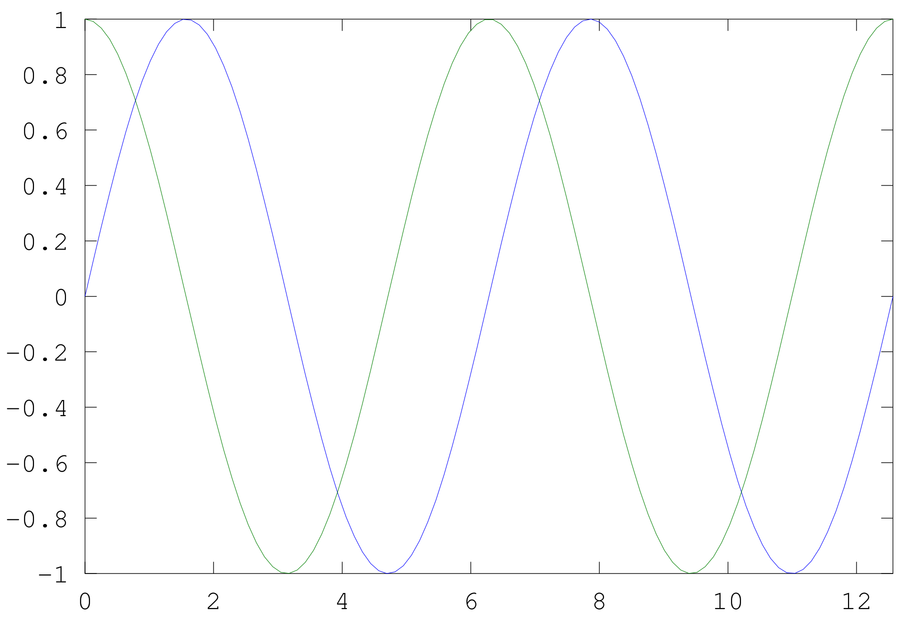
For instance, here's how to do a basic function plot. We have to import the `gnuplot` module and, since we're doing all calculations in Pure here, we also need the `math` module for the `sin` and `cos` functions. Also note that the Octave plotting functions are all in their own `gnuplot` namespace, so for convenience we import this namespace. The `gnuplot` module also provides various utility functions such as `linspace` which we use here to generate a vector of x values for the plot.

> **using** gnuplot, math; **using namespace** gnuplot;

> **let** $x = \text{linspace}(0, 4\,\pi)$; **let** $\text{fig1} = \text{plot}(x, \text{map}\sin x, x, \text{map}\cos x)$; axis "tight";

```
()
```

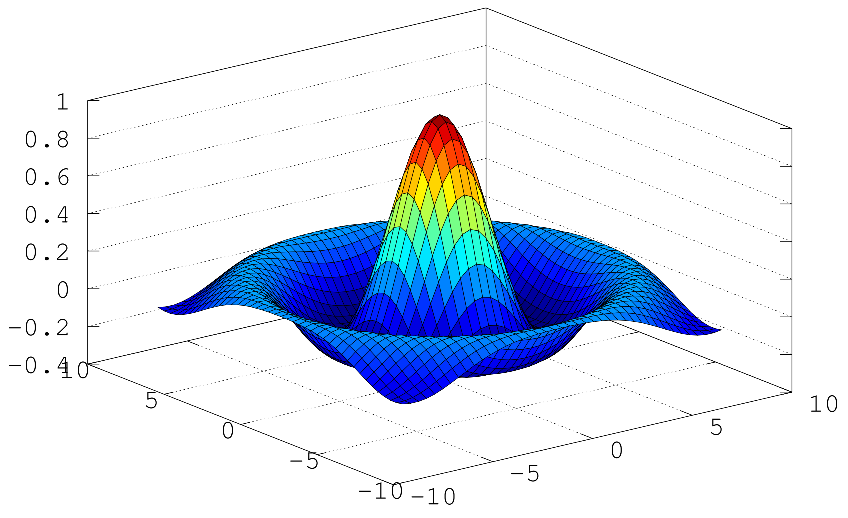
> psplot();



```
"/tmp/oct-7P9aFt.eps"
```

As you can see, the `psplot` function also returns the name of the PostScript file it generated, which is a temporary file by default. It's also possible to explicitly specify a filename and other options understood by the `gnuplot::print` command. For instance:

> **let** fig2 = sombrero(); psplot "sombrero.eps";



```
"sombrero.eps"
```

The given filename is interpreted relative to the current directory of the interpreter (which is normally the directory in which TeXmacs was started).

You can also display the Gnuplot window for 3d viewing, and hide it again, as follows (these operations aren't part of Octave's plot functions, but are provided as convenience functions in the `gnuplot` module):

```
> popup();
()
> popdn();
()
```

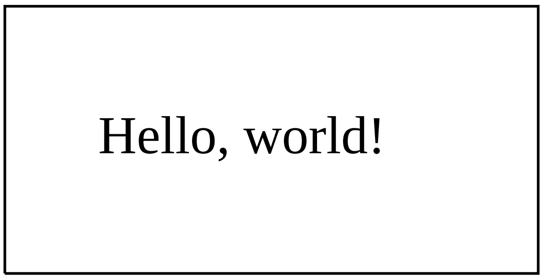
Octave's Gnuplot interface is very comprehensive. Please refer to the Octave manual for a description of the provided plotting functions. Most of the plotting examples given in the Octave documentation can be translated to Pure in a fairly straightforward manner. More examples can be found in the `pure-octave` package.

7. PostScript Output

Like the Python plugin, Pure can pipe PostScript graphics directly into TeXmacs. This is done with the `ps` function from the `texmacs` module which takes either verbatim PostScript code (indicated by a `%!` header) or a PostScript filename as its string argument. In the latter case, if the PostScript filename doesn't have a slash in it, it is searched for in all directories on the `TEXMACS_DOC_PATH`, and a `.eps` or `.ps` suffix is added automatically when needed. Otherwise the filename is interpreted relative to the current directory of the interpreter (which is normally the directory in which TeXmacs was started).

Here is a simple example illustrating verbatim PostScript code:

```
> ps "%!\n%%BoundingBox: 195 295 405 405\n/Times-Roman findfont 20 scalefont
  setfont newpath 200 300 moveto 200 0 rlineto 0 100 rlineto -200 0 rlineto 0 -
  100 rlineto 235 345 moveto (Hello, world!) show closepath stroke showpage";
```



```
()
```

8. Pure Scripting

Last but not least, Pure can also be used as a *scripting language* in $\mathrm{T_EX_{MACS}}$. This is done by enabling the Document | Scripts | Pure option (or Document | Scripts | Pure-math if you prefer math output; this is what we use here). This enables a few additional options in the Pure plugin menu, as well as the Insert | Link menu and the corresponding toolbar button. These facilities are described in more detail at the end of the "$\mathrm{T_EX_{MACS}}$ as an interface" section in the $\mathrm{T_EX_{MACS}}$ online help, so we only give a few basic examples here.

For instance, you can just select any valid Pure expression in the text and evaluate it with the Evaluate option of the Pure menu or `Ctrl+Return` like this: $(a+b)^2 \rightarrow a^2+2\,a\,b+b^2$. It's also possible to create an *executable switch* such as Pure-script-math $\mathrm{df}((x+y)^3, x)$ which can be toggled between input and computed result by pressing `Return` inside the field (try it!).

Note that as an additional convenience, the scripting plugins accept a simple expression without the trailing semicolon as input. This is in contrast to the regular Pure plugins which allow you to enter definitions and expressions spanning multiple input lines, but also require you to terminate each input item with a semicolon. With the scripting plugins the terminating semicolon is optional and will be added automatically when needed, but it also doesn't hurt if you type it anyway: Pure-script-math $\mathrm{df}((x+y)^3, x);$

There's also the possibility to work with *executable fields* and *spreadsheets*. These offer the advantage that fields may depend on other fields in the same document.[8] For instance, here is an example of a textual spreadsheet (Insert | Table | Textual spreadsheet) showing some Pure and Reduce calculations. Type `Return` in the cells of the last column to reveal the underlying Pure formulas; also try changing some of the values in the `b` and `c` columns and hitting `Return` to recompute the corresponding values in the last column.

a1	b	c	d
2	1	12	1932053504
3	17	33	50
4	$\sin(x^2)$	x	$2\cos(x^2)\,x$
5	$(x+y)^3$	x	$3\,(x^2+2\,x\,y+y^2)$

You can also refer to a table field in text like this: $2\cos(x^2)\,x$. Here we set the Ref field of the table to `table1`; the reference to cell `d4` can then be entered with the following series of key strokes: `\!\?` `table1-d4` `Return`.

Executable fields can be used to run arbitrary Pure code. For instance, here's the PostScript example from the previous section as a field (again, you can hit `Return` on the field to reveal the actual Pure code):

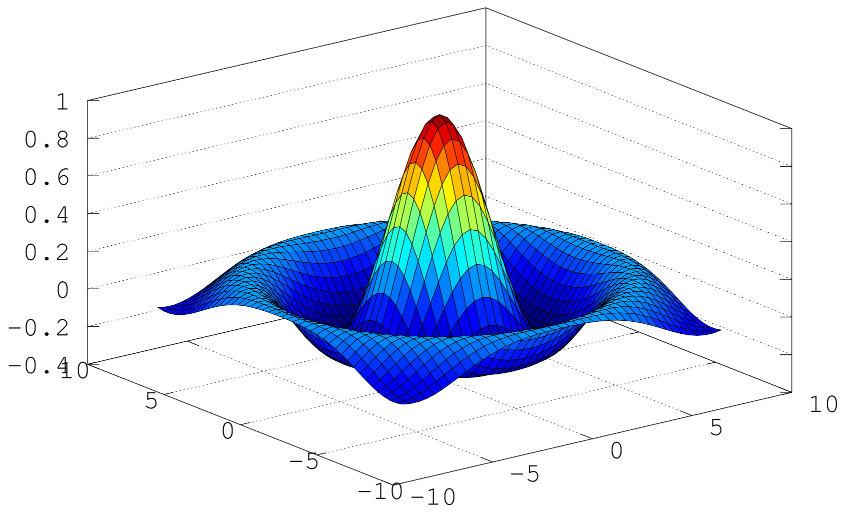
8. Note that the spreadsheet functionality requires $\mathrm{T_EX_{MACS}}$ 1.0.7.15 or later to work. Also be warned that, at least at the time of this writing, this might become *very* slow in large documents; however, it's possible to work around this limitation by breaking your document into smaller include files.

Hello, world!

Note that scripting by default uses its own instance of the Pure interpreter which is separate from all Pure sessions that might be included in the same document. But with executable switches it's possible to hook into any Pure session, so that you can use variables and functions defined or imported in that session. To do this, you'll have to create an executable switch for a specific session type (Insert | Fold | Executable). (Also note that in this case the expressions *must* be terminated with a semicolon, just like in a regular Pure session.) For instance, the following switch hooks into the Octave session from above:

$$\begin{pmatrix} -0.824564840132 & -0.415973557919 \\ 0.565767464969 & -0.909376709132 \end{pmatrix}, \begin{pmatrix} -0.372281323269 & 0 \\ 0 & 5.37228132327 \end{pmatrix}$$

In the same fashion we can also reproduce the plot of the sombrero:


1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
10
5
0
-5
-10
-10
-5
0
5
10