The eval symbol for axiomatising variadic functions

The list4 content dictionary

Lars Hellström
lars.hellstrom@residenset.net

Mälardalen University

Conference on Intelligent Computer Mathematics 2014, Coimbra
Summary

• In OpenMath, you’re allowed to have variadic functions/operations.

• It is not obvious how to state Formal Mathematical Properties that cover the infinitely many syntactical forms that a variadic function may have.

• But utilities to this end can be added as symbols in content dictionaries, notably the proposed list4#eval.
The Tcl language

- Originally designed to provide a scripting layer (somewhat in the spirit of Unix shell languages) within existing applications. Has since evolved into a general purpose programming language.

- Operates primarily as a dispatch engine. Source code is parsed according to a very simple core syntax. Once a complete command sentence has been scanned and all the words of that been resolved to explicit values, control is passed to a custom command procedure based on the value of the first word. The command procedure receives the complete list of argument values, as in the Unix argv model, but knows not what the source code looked like.
The Tcl language (2)

The core syntax includes:

- Read-access of variable ($\langle\text{varname}\rangle$).
- Use result of a command ([\langle\text{command sentence}\rangle]).

The core syntax does not include:

- write-access to variable (use \texttt{set} or some other command),
- arithmetic (use \texttt{expr} command or some ::\texttt{tcl}::\texttt{mathop} command),
- control structures (\texttt{if}, \texttt{while}, \texttt{for}, etc. are syntactically ordinary commands),
- pattern matching (a command procedure may do this, however).

This division of responsibilities is similar to that of OpenMath.
Variadic commands and their description

In Tcl, variadic commands are quite common. Some reasons are:

- Basic operational variadicity (like `arith1#plus` or `list1#list`).
- Options (‘long option’ syntax preferred).

The practice in Tcl is much wilder than what is reasonable in OpenMath, so a system sufficient for Tcl should be sufficient also for OpenMath.

What seems sufficient is to distinguish between two ways in which an entity can be part of a command sentence / `argv` vector:

- as one distinct item, or
- as a (possibly empty) sequence of items.

One of the main shortcomings of ordinary Unix shell languages for programming is that they have trouble keeping these apart.
To splode or not to splode

Tcl has one syntax for specifying the difference: the expand operator \{\star\}, also known as the *splode*.

- If a word does not begin with a splode, it contributes exactly one element to the \textit{argv} vector.
- If a word begins with a splode, the rest of the word is interpreted as a value as usual. Then that value is interpreted as a list, and the elements of that list are added as elements of the \textit{argv} vector.

This meets a need similar to that of the HKR sequences proposal, but with fewer primitives since lists (sequences) are just another class of values.

\[
\begin{align*}
\langle \text{OMSV} \rangle & \quad \text{corresponds to} \quad \text{splode} + \langle \text{OMV} \rangle \\
\langle \text{OMSEQ} \rangle & \quad \text{corresponds to} \quad \text{splode} + \langle \text{OMA} \rangle
\end{align*}
\]
The Tcl splode is a “new” feature

The Tcl core syntax is a rather conservative one. Apart from some escapes for Unicode characters, the splode is the *only* new syntax feature to have been introduced during the last 20 years. And even when it was introduced (2003), it was controversial.

One argument *against* it was that it technically wasn’t needed: anything you could do using the splode could also be done using the `eval` command. Instead of

```
glob -dir $mydir {[*]}$filePatternList
```

you could write

```
eval [list glob -dir $mydir] $filePatternList
```
**Tcl eval and list4#eval**

Functionality-wise, the Tcl `eval` allows for evaluating dynamically generated code. Implementation-wise, it is just a recursive call back into the dispatch engine.

The `list4#eval` symbol aims to provide similar functionality in OpenMath, but here one primarily needs to state axioms:

```
list4.eval([$f]) = $f()
list4.eval([$f $x]) = $f($x)
list4.eval([$f $x $y]) = $f($x,$y)
list4.eval([$f $x $y $z]) = $f($x,$y,$z)
```

etc.
Not a complete solution

The `list4#eval` symbol allow you to express in OpenMath many things that involve sequences, including all Examples 1–3 that are in the HKR *Extending OpenMath with Sequences* paper.

I would not call it a complete solution, however, since variable bindings are not covered.

(Tcl does variable binding as a semantic rather than syntactic operation, so it is not really an issue there.)

On the other hand, `list4#eval` can probably express 90–95% of all variadic things you encounter in practical mathematics. It might not do it *elegantly*, but it can do it *here and now*. 