REDUCE functions provided by redcas

Martin Gregory

November 2, 2026

Contents

1 Introduction **1**

2 Extracting the REDUCE code **1**

3 Output procedures **2**
3.1 asltx: calls arrayltx or exprltx depending on the object type 2

4 Utility procedures **2**
4.1 array2flatls: converts an array to a flattened list 2
4.2 asltx_marker: mark output for extraction 3
4.3 atoi: converts a string to an integer 3
4.4 itoa: converts an integer to a string 3
4.5 lisp_dialect 3
4.6 swget 4
4.7 swreset 4
4.8 swset 4
4.9 swtoggle 4

5 Symbolic procedures **4**
5.1 arrayp: array predicate 5
5.2 concat: concatenate 2 strings 5
5.3 gettype: distinguishing arrays from "simple" expressions 5
5.4 id2string 5
5.5 onoff: required by swtoggle 5
5.6 tex_string: render string as LaTeX 6

1 Introduction

The *redcas* package provides a number of REDUCE procedures which are required for the package to function. While knowledge of these procedures is not needed to use *redcas*, they may be of interest for use in REDUCE itself. There are three types of procedures. The first type produce outputs in a format which can be used by *redcas*. The second type are utility procedures needed by the first type. Finally there are declarations of symbolic procedures for use in algebraic mode using the `symbolic operator` statement. This third type are also utilites used by the first type.

2 Extracting the REDUCE code

All procedures are included in the file *redcas.red* in the *reduce* directory of the installed package. You can extract the code to a location of your choice using the following code:

```
file.copy(system.file("reduce/redcas.red", package="redcas"), "mydir")
```

where *mydir* is a writable directory of your choice.

3 Output procedures

3.1 asltx: calls arrayltx or exprltx depending on the object type

Syntax `asltx(x, math, mode, name) ;`
`arrayltx (arrx, math, mode, name) ;`
`exprltx(expr, math, mode, name) ;`

where

x identifier of an object to be typeset.

arrx identifier of an array.

expr an expression or identifier of such.

math string naming a LaTeX math environment to enclose the expression or each element of an array. If empty no math environment is written.

mode string naming the print mode: nat|fancy. *nat* is the standard REDUCE plain text output while *fancy* produces LaTeX output using the REDUCE package TMPRINT. If not specified, defaults to nat.

name string providing the name to use when printing.

Details *asltx* is a convenience function which calls *arrayltx* for arrays, *exprltx* for expressions and returns an error for any other type.

arrayltx accepts an array of arbitrary dimensions and displays each element using the specified mode.

exprltx prints an expression using the specified mode. Expressions may be unbound identifiers or those containing an expression, equation or matrix. Explicit expressions may also be specified.

Since REDUCE procedures do not support named arguments[1], all arguments must be specified.

Plain text has two modes controlled by the NAT switch. When on, the default, expressions will print on multiple lines if they include exponents or fractions. When off, expressions print on a single line in the form expressions are input and include as terminator the dollar sign.

Since there is a print method for matrices a math mode will enclose the entire matrix, not the elements.

Value None. Called for side effect of producing the output in the desired format.

4 Utility procedures

4.1 array2flatls: converts an array to a flattened list

array2flatls converts an array to a flattened list using the procedure *array_to_list* from the REDUCE package ASSIST and a for loop using the *join* action. This is called by *arrayltx* to allow handling arrays with an arbitrary number of dimensions.

Syntax `array2flatls(arrx) ;`

Arguments **arrx** identifier of array to convert

Value A list containing the elements of the array. Assuming the array has n dimensions the list will have depth n and the array element $(i_1, \ldots, i_n)$ is found by descending the list in the order of i_j. In terms of the dot operator from the ASSIST package and assuming the list is named l, this would be $(((l.i_1).i_2).\ldots).i_n$ or in R notation $l[[i_1]][[i_2]]\ldots[[i_n]]$

[1] lists might be used

4.2 asltx_marker: mark output for extraction

asltx_marker calls *asltx* and encloses the output between a line `##START label` and `##END label`, so that it can be easily extracted from a REDUCE log.

Syntax `asltx_marker (thing, math, mode, name, label) ;`

Arguments **x** identifier of an object to be typeset.

math string naming a LaTeX math environment in which to enclose each array element. If empty no math environment is written.

mode string naming the print mode: nat|fancy. *nat* is the standard REDUCE output while *fancy* produces LaTeX output using the REDUCE package TMPRINT. If not specified, defaults to nat.

name string providing the name to use when printing expressions.

label an arbitrary string to identify the output.

Details This procedure allows extraction of specific outputs from the log of a reduce program which has been run independently of *redcas*

Value None. Called for side effect of producing the output in the desired format enclosed in the start and end markers. This function is used by some package tests to extract output from REDUCE programs executed outside *redcas*.

4.3 atoi: converts a string to an integer

atoi converts a string containing an integer or float in base 10 to a number.

Syntax `atoi(string) ;`

Arguments **string** a string containing only base 10 digits and a single period.

Value a number. ROUNDED must be off in order to convert a float.

4.4 itoa: converts an integer to a string

itoa converts an integer to a string. Useful for formatting output.

Syntax `itoa(integer) ;`

Arguments **integer** an arbitrary integer.

Value a string representation of the integer.

4.5 lisp_dialect

lisp_dialect determines whether REDUCE is running under CSL or PSL

Syntax `lisp_dialect() ;`

Arguments None

Details this function is intended for use in a condition, for example

```
if lisp_dialect = 'csl then ... ;
```

Value the quoted symbol `'csl` or `'psl`

4.6 swget

swget tests whether a REDUCE switch is on or off.

Syntax `swget(s)`

where

s the identifier of the switch to test

Details if the value of *swget* is *nil* it prints as blank. It should only be used in a condition, for example,

```
if swget(echo) then write "on" else write "off";
```

Value Boolean.

4.7 swreset

swreset restores the status of switches which were toggled by *swset*.

Syntax `swreset(l)`

where

l the identifier of the list of toggled switches produced by *swset*.

Details *swreset* uses *swtoggle* to restore the previous state and displays the restored state

Independently of *swset* an explicit list may also be passed in order to toggle multiple switches.

Value None.

4.8 swset

swset ensures that one or more switches are in a desired state.

Syntax `swset(lon, loff)`

where

lon is a list, or identifier of the list, of switches which should be on.

loff is a list, or identifier of the list, of switches which should be off.

Details *swset* checks the current state of each switch in *lon* and *loff* and if it is not in the desired state uses *swtoggle* to change it.

Value A list of the switches which were toggled. This can be later passed to *swreset* in order to restore the original state.

4.9 swtoggle

swtoggle toggles a REDUCE switch.

Syntax `swtoggle(s) ;`

Arguments **s** the identifier of the switch to toggle

Value the new state of the switch.

5 Symbolic procedures

This section describes existing symbolic procedures which have been declared algebraic by using the `symbolic operator` statement.

5.1 arrayp: array predicate

arrayp is a predicate function to test whether an object is an array or not.

Syntax `array(x)`

Arguments **x** an identifier to test

Details if the value of *arrayp* is *nil* it prints as blank. It should only be used in a condition, for example,

```
if arrayp(x) then ... ;
```

Value Boolean.

5.2 concat: concatenate 2 strings

concat concatenates two strings.

Syntax `result := concat(a, b) ;`

Arguments **a** first string

b second string

Details

Value string containing the concatenation of the first and second strings.

5.3 gettype: distinguishing arrays from "simple" expressions

gettype returns the type of an identifier.

Syntax `gettype(x)`

Arguments **x** the object for which type should be returned

Details *gettype* is used by *asltx* to determine whether to call *arrayltx* or *exprltx*.

Value the type of the object as a quoted identifier, for example, `'array`.

5.4 id2string

id2string returns an identifier's name as a string

Syntax `id_as_string := id2string(x) ;`

Arguments **x** identifier whose name is to be returned as a string

Value string containing the name of the identifier.

5.5 onoff: required by swtoggle

onoff is a symbolic function which sets a switch.

Syntax `onoff(s, bool) ;`

Arguments **s** identifier of the switch.

bool a boolean to set the switch on (t) or off (nil).

Details This is used by swtoggle.

Value None, used for side effect of changing switch state.

5.6 tex_string: render string as LaTeX

tex_string prevents REDUCE replacing \ with \textbackslash and {} with \{\} when the FANCY switch is on.

Syntax `result := tex_string(s) ;`

Arguments **s** a string to render as LaTeX

Details *redcas* uses *tex_string* when writing the math environment to ensure the string is not modified.

Value the original string without the unwanted conversions.