Package 'redcas'

November 8, 2024

Type Package

Depends R (>= 3.5.0), methods

Title An Interface to the Computer Algebra System 'REDUCE'

Version 0.1.1

Date 2024-11-05

Author Martin Gregory [aut, cre]

Maintainer Martin Gregory <mairtin.macghreagoir@gmail.com>

Description 'REDUCE' is a portable general-purpose computer algebra system supporting scalar, vector, matrix and tensor algebra, symbolic differential and integral calculus, arbitrary precision numerical calculations and output in 'LaTeX' format. 'REDUCE' is based on 'Lisp' and is available on the two dialects 'Portable Standard Lisp' ('PSL') and 'Codemist Standard Lisp' ('CSL'). The 'redcas' package provides an interface for executing arbitrary 'REDUCE' code interactively from 'R', returning output as character vectors. 'R' code and 'REDUCE' code can be interspersed. It also provides a specialized function for calling the 'REDUCE' feature for solving systems of equations, returning the output as an 'R' object designed for the purpose. A further specialized function uses 'REDUCE' features to generate 'LaTeX' output and post-processes this for direct use in 'LaTeX' documents, e.g. using 'Sweave'.

License GPL-3

SystemRequirements 'REDUCE' <http://www.reduce-algebra.com>. At least one of the 'CSL' or 'PSL' dialects must be installed in order to use the package. Tested with 'REDUCE' releases 6860 and 6558 for both 'CSL' and 'PSL' on Linux, and 6558 and 6339 for 'CSL' on MacOS

NeedsCompilation no

OS_type unix

Repository CRAN

Date/Publication 2024-11-08 14:50:02 UTC

Contents

redcas-package 2

asltx . . . 3
print . . . 7
redcas.solve . . . 9
redClose . . . 10
redCodeDir . . . 11
redDropOut . . . 12
redExec . . . 13
redExpand . . . 15
redSolve . . . 16
redSplitOut . . . 19
redStart . . . 20
showSessions . . . 22

Index **24**

`redcas-package` *An Interface to the Computer Algebra System 'REDUCE'*

Description

'REDUCE' is a portable general-purpose computer algebra system supporting scalar, vector, matrix and tensor algebra, symbolic differential and integral calculus, arbitrary precision numerical calculations and output in 'LaTeX' format. 'REDUCE' is based on 'Lisp' and is available on the two dialects 'Portable Standard Lisp' ('PSL') and 'Codemist Standard Lisp' ('CSL'). The 'redcas' package provides an interface for executing arbitrary 'REDUCE' code interactively from 'R', returning output as character vectors. 'R' code and 'REDUCE' code can be interspersed. It also provides a specialized function for calling the 'REDUCE' feature for solving systems of equations, returning the output as an 'R' object designed for the purpose. A further specialized function uses 'REDUCE' features to generate 'LaTeX' output and post-processes this for direct use in 'LaTeX' documents, e.g. using 'Sweave'.

Details

The package allows you to start a REDUCE session (`redStart`), send one or more vectors of REDUCE commands to the session for execution and retrieve the output (`redExec`), render output as LaTeX (`asltx`), and shut down the session and optionally return the entire output file (`redClose`). Output is returned as a list of character vectors containing the command output, the commands and the combined comands and output. `redcas` can also directly call the REDUCE operator for solving algebraic equations (`redSolve`), returning a `redcas.solve` object.

Several other functions are available: `redExpand` is called by `redExec` for checking and preparing the input vector while `redDropOut` and `redSplitOut` are called by `redExec` to format the output. All three can be called independently of `redExec`.

REDUCE uses a different terminology to R. Some translations:

REDUCE	**R**
command:	statement
log:	transcript

operator (prefix):	function
operator (infix):	operator
procedure:	function
switch:	(global) option

Author(s)

Martin Gregory [aut, cre]

Maintainer: Martin Gregory <mairtin.macghreagoir@gmail.com>

References

Details of the REDUCE system including extensive documentation is available at the REDUCE web site.

Examples

```
## start a session
id <- redStart()

## can only run code if session was successfully started
if (is.numeric(id)) {
   ## define vectors of REDUCE code for submission
   x1 <- "solve((x+1)^2, x);"
   x2 <- "b:=factorize(45056);"
   x3 <- c("operator v; ",
   "la:=-m/(v(2)^2 + v(3)^2)^(1/2) ;")

   ## execute x1 and display on the output of the commands
   y1 <- redExec(id, x1)
   writeLines(y1$out)

   ## execute x2 and display first the output and then the commands
   y2 <- redExec(id, x2)
   writeLines(c(y2$out, y2$cmd))

   ## execute x3 and display the log (commands and output interspersed)
   writeLines(y3 <- redExec(id, x3, split=FALSE))

   ## close the session
   redClose(id)
   }
```

---

asltx *Convert REDUCE output to LaTeX*

Description

sends the name of an array or expression to the REDUCE process for display either as is or as LaTeX using the FANCY switch. If the object is an array, each element is printed. The LaTeX produced by REDUCE is then post-processed to translate object names according to a map specified by the user, to convert array arguments to indices and, optionally, to enclose the result in a math environment specified by the user.

Usage

```
asltx(id, x, usermap, mathenv="", mode="fancy", notify=0, timeout=0, debug=FALSE)
```

Arguments

`id`	session identification returned by `redStart`. Required.
`x`	A character vector with one element naming the object to display.
`usermap`	A list containing one or both of the following items: **ident** a named character vector specifying translation of REDUCE identifiers to LaTeX names. The element name is the REDUCE identifier and the value the LaTeX name. **index** a named character vector specifying the conversion of arguments to indices. The name specifies which identifier to process. Since this mapping is done after the identifier mapping, you need to use the LaTeX name for any name you have mapped. The value is a sequence of `^` and `_` characters specifiying whether the index is contravariant (superscript) or covariant (subscript), respectively. This argument is optional. Default is no user-specified translation.
`mathenv`	Character. The name of a LaTeX math environment in which to enclose an expression or each element of an array. Defaults to no environment. This is useful for displaying the array elements in a subordinate math environment, for example `\dmath*` within a `dgroup` environment.
`mode`	Character. Whether to display `x` as LaTeX (`"fancy"`) or not (`"nat"`). Default is `"fancy"`. `""` is treated as `"nat"`.
`notify`	while waiting for the REDUCE commands to complete, write a note to the console every `notify` seconds. Default is zero which suppresses the message.
`timeout`	numer of seconds after which to terminate the function if it is still waiting for output from the REDUCE process. Default is 0 which will never initiate termination.
`debug`	Boolean. When TRUE the mappings are printed to enable debugging since regular expressions can be tricky. Default is FALSE.

Details

asltx uses `redExec` to display the object `x` as desired by:

1. constructing a call to the REDUCE function `asltx` with appropriate quoting of the arguments;
2. executing this call using redExec and specifying `split=TRUE`;

3. applying a standard set of transformations to remove some non-LaTeX markup and perform some conversions on the result. See below;
4. applying a set of transformations specified by the `usermap` argument.

Features of REDUCE conversion to LaTeX: REDUCE provides three different methods for converting output to LaTeX, the packages TMPRINT, RLFI and TRI. **redcas** uses the TMPRINT package which was designed for use with the TeXmacs editor (hence TM) and triggers conversion using the `fancy` and `fancy_tex` switches. TMPRINT was chosen because it can be easily applied to fragments, supports using the `\frac` command, produces LaTeX output and is well supported. In contrast RLFI is designed to produce complete documents using LaTeX 2.09 syntax and does not support the `\frac` command. TRI produces plain TeX output.

TMPRINT supports

1. inserting `\left` and `\right` in nested parentheses, braces and brackets;
2. converting variables whose names are those of Greek letters to the corresponding LaTeX command. Capitalized names are mapped to the command for the capital. For example, `psi` is mapped to `\psi` while `Psi` is mapped to `\Psi`. There are two exceptions; `epsilon` and `kappa` are mapped to `\varepsilon` and `\varkappa`, respectively. If you need `\epsilon` or `\kappa` you can use the `usermap` argument;
3. the following names are mapped to special symbols:

infinity	\infty	union	\cup
partial!-df	\partial	member	\in
empty!-set	\emptyset	and	\wedge
not	\neg	or	\vee
not	\neg	when	|
leq	\leq	!*wcomma!*	,\,
geq	\geq	replaceby	\Rightarrow
neq	\neq	!~	\forall
intersection	\cap		

Standard transformations: Standard transformations done by `asltx` are as follows:

1. removal of markup used for interactive display in TeXmacs and which has no correspondence with LaTeX markup;
2. replacement of REDUCE assignments `:=` and `~:=~` with a plain equals sign;
3. reversing the unwanted conversion of numeric suffix to index. The REDUCE FANCY option assumes that a numeric suffix to a variable is a subscript index even when the variable is an array, e.g. `cs1(a,b)` is converted to `cs_{1}(a,b)`. `asltx` undoes this conversion.
4. removal of redundant `mathrm` commands;
5. removal of unnecessary trailing spaces from Greek capitals;
6. replacement of `mathit{Q}` with `mathrm{Q}` where `Q` is one of the Greek capitals A, B, E, Z, H, I, M, N, O, P. i.e. those which have the same form as a Latin capital. This transformation is required to have the same type for as the Greek-only capitals.

User transformations:

The `usermap` argument allows renaming variables, converting arguments in REDUCE objects to indices and specifying whether indices are covariant or contravariant. If a LaTeX command is

used in the map, four backslashes are required to get the single backslash for LaTeX because the mapping is done using regular expressions. For example suppose we have arrays `s`, `g`, `kminus` and `kplus` in REDUCE and want the following mapping to LaTeX

REDUCE	LaTeX
s	\Sigma
g	g
kminus	K^-
kplus	K^+

then the `ident` element of the list passed to `usermap` would be

```
ident=c("s"="\\\\Sigma", kminus="K^-", kplus="K^\\+")
```

Since we are not mapping `g` it does not need to appear here.

To illustrate mapping arguments to indices, assume that all four arrays in the example above are two dimensional and `s` has one subscript and one superscript, `g` and `kminus` have two subscripts and `kplus` has two superscripts then the `index` element will be

```
index=c("\\\\Sigma"="_^", "g"="__", "K^-"="__", "K^\\+"="^^")
```

Note that the name of the `index` elements must be the result of applying the `ident` mapping. The names are regular expressions so the character + must be escaped in the names.

Value

A list containing the following elements is returned:

tex the transformed LaTeX output;

out the output of the executed commands;

cmd the executed commands

raw the interspersed commands and output.

Author(s)

Martin Gregory

See Also

`redExec` for details of executing REDUCE code.

Examples

```
## start the session
s1 <- redStart()

## can only run code if session was successfully started
if (is.numeric(s1)) {
   ## create the arrays
   redcode <- c("array g(2,2), s(3,3), kplus(2,2), kminus(2,2) ;",
```

```
"operator x;",
"g(0,0) := -u^(-2);",
"g(1,1) := (u*x(3))^2;",
"g(2,2) := g(1,1) * (sin(x(1)))^2;",
"s(0,0) := 0;",
"s(1,1) := df((u*x(3))^2, x(3));",
"s(2,2) := df(g(2,2), x(1));",
"s(3,3) := u^2;",
"kplus(0,0) := df(g(0,0), u);",
"kplus(1,1) := df(g(1,1), u);",
"kplus(2,2) := df(g(2,2), u) ;",
"kminus(0,0) := df(g(2,2), u);",
"kminus(1,1) := df(g(1,1), u);",
"kminus(2,2) := df(g(0,0), u) ;",
"on nero ;"
)

   o2 <- redExec(s1, redcode)

   ## create LaTeX output
   writeLines(c("", asltx(s1, "g", mathenv="")[["tex"]], ""))
   writeLines(c("", asltx(s1, "s", mathenv="")[["tex"]], ""))
   writeLines(c("", asltx(s1, "kplus", mathenv="")[["tex"]], ""))
   writeLines(c("", asltx(s1, "kminus", mathenv="")[["tex"]], ""))

   ## close the session
   redClose(s1)
}
```

`print` *Print a redcas.solve object*

Description

`print.redcas.solve` is a print method for function `print` in package **base**

Usage

```
print(x, ...)
```

Arguments

`x`	a `redcas.solve` object returned by the function `redSolve`.
`...`	other arguments passed to print.

Details

`print.redcas.solve` displays the equations, solutions if any and the list of unknowns and switches. The solutions displayed are those returned from REDUCE without any transformation to R objects.

The layout depends on the length of the individual solution values and the `width` option. If there is sufficient space, the solutions for each unknown are printed in separate columns on the same line, for example with a width of 80, the output of the example below will be:

```
Equations:
  x+y+z = 0
  x^2 + y^2 + z^2 = 9
  x^2 + y^2 = z^2

Number of solutions: 4

Solutions:
               x               y               z
   2.12132034356               0  -2.12132034356
  -2.12132034356               0   2.12132034356
               0   2.12132034356  -2.12132034356
               0  -2.12132034356   2.12132034356

Unknowns: x,y,z
Switches:  rounded
```

If all values for a solution with not fit on a line, they are printed below each other. With a width of 30, the solutions section would be

```
Solution 1:
x:    2.12132034356
y:                0
z:   -2.12132034356

Solution 2:
x:   -2.12132034356
y:                0
z:    2.12132034356

Solution 3:
x:                0
y:    2.12132034356
z:   -2.12132034356

Solution 4:
x:                0
y:   -2.12132034356
z:    2.12132034356
```

Value

`print.redcas.solve` is called for its side-effect and returns NULL.

Examples

```
## start the session
r0 <- redStart()

## can only run code if session was successfully started
if (is.numeric(r0)) {
   rsobj <- redSolve(id=r0,
     eqns=c("x+y+z = 0", "x^2 + y^2 + z^2 = 9", "x^2 + y^2 = z^2"),
     unknowns=c("x", "y", "z"),
     switch=c("on rounded;") )
   print(rsobj)
   redClose(r0)
}
```

---

redcas.solve *Class* "redcas.solve"

Description

Contains the inputs and outputs of a call to the function `redSolve` which executes the REDUCE `solve` operator to determine solutions to a system of algebraic equations.

Usage

```
redcas.solve(...)
```

Arguments

`...`	see section Slots for details.

Objects from the Class

Objects are created by calls to the function `redSolve` and can be used as input to the `print` method.

Slots

`.Data`: Object of class `"list"` containing the actual data.

`solutions`: Object of class `"list"` containing a character vector for each solution of the system of equations. The character vectors have a named element for each unknown with the name being the unknown and the value being the value of the unknown for the solution. This is essentially a copy of the list returned by the REDUCE `solve` operator.

`rsolutions`: Object of class `"list"` containing a list for each solution converted to appropriate R types where possible. A vector will not suffice because the values of the unknowns may be of different types: integer, real, complex or expression. Explicit real and complex numbers are always converted. If the REDUCE switch `ROUNDED` is off, real and complex numbers may be expressions and may result in errors or NaN if conversion were attempted.

`nsolutions`: Object of class `"numeric"`. The number of solutions

`rc`: Object of class `"numeric"`. Return code from the `redSolve` call: 0 is success and 1 failure.

`root_of`: Object of class `"complex"`. A solution may express one or more unknowns in terms of the roots of another equation. In this case the other equation is enclosed in `root_of()`. This slot is a complex vector with the real part identifying the solution contain `root_of()` and the imaginary part the index of the unknown in that solution.

`eqns`: Object of class `"character"`. The system of equations provided to `redSolve`

`unknowns`: Object of class `"character"`. The unknowns provided to `redSolve`

`switches`: Object of class `"character"`. Any REDUCE switches set prior to executing the REDUCE `solve` function.

Extends

Class `"list"`, from data part. Class `"vector"`, by class "list", distance 2.

Methods

A print method, `print`, is defined.

Author(s)

Martin Gregory

See Also

`redSolve` is currently the only function using this class.

Examples

```
showClass("redcas.solve")
```

---

redClose *function to close a REDUCE session and optionally save the log file*

Description

`redClose` closes the connection to the REDUCE session thereby ending the session. If requested the log file is copied from the temporary location to one specified by the `log` argument.

Usage

```
redClose(id, log)
```

Arguments

id	the session identifier returned by redStart. Required with no default
log	the path to the location in which to save the session's log file. Optional.

Details

The session log returned by redClose includes the submit block markers set by redExec.

In addition to closing the connection, redClose removes the session's entry from the session registry.

Value

redClose returns TRUE if the session is closed, FALSE if it does not exist.

Author(s)

martin gregory

See Also

redStart for creating a REDUCE session and showSessions to display the session registry.

Examples

```
## Open a CSL session:
s1 <- redStart()
## can only run code if session was successfully started
if (is.numeric(s1)) {
   ## show session details:
   print(showSessions())
   ## close session:
   redClose(s1)
}
```

---

redCodeDir *Retrieve the directory containing the REDUCE code which is part of* redcas.

Description

redCodeDir retrieves the full path to the directory containing the REDUCE code which is part of redcas.

Usage

```
redCodeDir()
```

Details

In addition to returning the REDUCE code directory, `redCodedir` assigns three environment variables:

REDCODEDIR the full path to the directory containing the REDUCE code which is the `reduce` sub-directory of the `redcas` package;

REDCAS_CODE_PATH the full path to the program `redcas.red` which defines REDUCE procedures used by `redcas`;

TMPRINT_PSL_PATH the full path to the program `tmprint-psl.red` which makes a modified version of the REDUCE package TMPRINT which implements LaTeX rendering for the PSL dialect. The change is minor and is designed to ensure that the TeXmacs markup is identical to that of CSL.

You can use the two programs directly in REDUCE. You need only load `redcas.red` as it loads `tmprint-psl.red` if executing in PSL. In order to have `tmprint-psl.red` loaded, you must assign the environment variable `TMPRINT_PSL_PATH` before starting REDUCE.

Value

`redCodeDir` returns the full path to the directory containing the REDUCE code which is part of `redcas`.

Author(s)

martin gregory

Examples

```
redCodeDir()
Sys.getenv("REDCAS_CODE_PATH")
Sys.getenv("TMPRINT_PSL_PATH")
```

`redDropOut` *Removes certain elements from the REDUCE output.*

Description

Removes blank lines or the submit block end marker from the REDUCE output.

Usage

```
redDropOut(x, what)
```

Arguments

`x`	a character vector containing REDUCE output
`what`	a character string describing what to drop. `what='blank'` removes blank lines, i.e. those matching regular expression "^$" while `what='marker'` removes the end of block marker

Details

redDropOut is called by redExec if the option drop.blank.lines is TRUE, so you generally do not need to call it explicity. If you have forgotten to set the option, you can use this function to remove blank lines.

The purpose of what='marker' is to remove the end of block marker described in the details section of redExec. This may be useful if you need to process a REDUCE transcript not produced by redcas.

When removing blank lines and the output is not LaTeX or the switch NAT is on, output may be difficult to read and difficult to parse.

Value

The input vector with the corresponding elements removed.

Author(s)

Martin Gregory

See Also

redExec

Examples

```
x <- c("first", "", "last")
## this will return elements 1 and 3:
redDropOut(x, "blank")
```

---

redExec *execute REDUCE commands from R*

Description

sends a vector of commands to the REDUCE process for execution and retrieves and formats the output.

Usage

```
redExec(id, x, split = TRUE, drop.blank.lines = FALSE, notify=0, timeout=0)
```

Arguments

id	session identification returned by redStart. Required.
x	A character vector contain either the REDUCE commands to execute, the names of REDUCE command files or a mixture of both. Required.
split	Logical. Should the output be split into separate vectors for echoed commands and for output. Default TRUE.

`drop.blank.lines`
: Logical. Should blank lines be removed from the output? Default FALSE.

`notify`
: while waiting for the REDUCE commands to complete, write a note to the console every `notify` seconds. Default is zero which suppresses the message.

`timeout`
: numer of seconds after which to terminate the function if it is still waiting for output from the REDUCE process. Default is 0 which will never initiate termination.

Details

Before submitting the vector `x`, `redExec` modifies it as follows:

- inserts a ";" terminator at the start of `x`. This is required to ensure that the first command in the next submit block has a number. Numbers are required to distinguish commands from command output;
- replaces any file names in `x` with their contents. This means that the contents of the files are executed directly and not via an `IN` or `IN_TEX` command. Files are specified by preceding the filename with `file:`. A terminator is not required and, if present, is ignored;
- appends a command to `x` in order to set an end of block marker. This is required so that `redExec` can wait until all the commands submitted have been executed. This is especially important for commands which run for a long time.

When reading the output from the submitted commands, `redExec` removes the end of block command and the marker it sets.

Care must be taken when using the `drop.blank.lines` option. In particular, if using the REDUCE `NAT` switch, which is the default, and output is not formatted for LaTeX, it may be difficult to read the output.

If REDUCE prompts for input during a sumbit block, `redExec` will write an informative message, return the output and close the session, writing the log to the current directory. This most likely happens if you use a variable as an operator but did not declare it as such. One way to avoid this is to turn off the switch INT which causes REDUCE to declare any undeclared operators it encounters. If you do this, be sure to turn on INT before you call `redClose` as otherwise the REDUCE session will hang. You should use the code `on int;;`. Note that both semicolons are required. Since REDUCE is run via a pipe, INT is on by default.

If the last statement (element of `x`) is missing a terminator (semi-colon or dollar sign) `redExec` checks whether the last non-blank, non-comment element of the input vector has a terminator. If not it stops with an appropriate message and returns FALSE. This reduces the likelihood that `redExec` fails to return.

Value

Like R, the REDUCE log contains both commands and outputs. If split is TRUE a list containing the following elements is returned:

out the output of the executed commands;

cmd the executed commands

raw the interspersed commands and output.

If split is FALSE, `redExec` returns a character vector with the output from the REDUCE commands.

Author(s)

Martin Gregory

See Also

redStart for starting a REDUCE session, redSplitOut and redDropOut for post-processing the output if the split or drop.blank.lines are not used.

Examples

```
s1 <- redStart()

## can only run code if session was successfully started
if (is.numeric(s1)) {
   ## submit with neither split nor drop.blank.lines
   x1 <- c("solve((x+1)^2, x);",
   "b:=factorize(45056);",
   "operator v; ",
   "la:=-m/(v(2)^2 + v(3)^2)^(1/2) ;")
   writeLines("## submit with neither split nor drop.blank.lines")
   writeLines(y1 <- redExec(s1, x1, split=FALSE))

   ## submit with split=TRUE and drop.blank.lines=TRUE
   writeLines("\n## submit with split=TRUE and drop.blank.lines=TRUE")
   writeLines(redExec(s1, x1, drop.blank.lines=TRUE)[["out"]])

   ## submit with file name in input
   tfile <- tempfile('reduce')
   writeLines(c(paste0('write "code from ', tfile, '";'),
"b:=factorize(54056);"),
      tfile)
   x3 <- c("la:=-m/(v(2)^2 + v(3)^2)^(1/2) ;",
   paste0("file:",tfile))
   y3 <- redExec(s1, x3, split=TRUE)
   writeLines("\n## submit with file name in input")
   writeLines("## commands")
   writeLines(y3$cmd)
   writeLines("\n## output")
   writeLines(y3$out)
   redClose(s1)
}
```

---

redExpand *Insert the content of a file into a vector*

Description

A vector of REDUCE commands may contain the name of a file containing REDUCE commands. redExpand returns a vector containing the commands up to the element containing the file name,

the contents of the file and the elements following the file name. Multiple files may be specified in the input vector.

Usage

```
redExpand(x)
```

Arguments

x	vector of commands to submit to REDUCE, possibly including names of files of commands to submit.

Details

redExpand is called by redExec before submitting commands so it is generally not necessary to call it explicitly. It can be used if you wish do the expansion yourself. If the input vector elements x_j, x_k, where $j < k$, are file references, then the output vector will consist of

- $x_1 \ldots x_{j-1}$
- the contents of x_j
- $x_{j+1} \ldots x_{k-1}$
- the contents of x_k
- $x_{k+1} \ldots x_n$

Value

a character vector containing the commands to submit with any file names replaced by their contents.

Author(s)

Martin Gregory

See Also

redExec

---

redSolve *function to solve a system of equations using the REDUCE* solve *operator*

Description

redSolve generates code for the REDUCE solve operator, executes it and converts the result to appropriate R objects if possible.

Usage

```
redSolve(id, eqns, unknowns, switches)
```

Arguments

id	the session identifier returned by redStart. Required with no default
eqns	the system of equations to be solved as character vector with one element per equation. If equations are in the normal form the =0 may be omitted. Required. No default.
unknowns	the unknowns for the system of equations specified by eqns as a character vector with one element for each unknown. Required. No default.
switches	a character vector of switches to be applied prior to executing solve. See Details below. Optional.

Details

solve is a REDUCE operator for solving one or more simultaneous algebraic equations in one or more variables, both linear and non-linear. Equations may contain arbitrary constants which is why it is necessary to specify the unknowns. For example, given a quadratic equation with constants a, b and c:

```
16: solve(a*x^2 + b*x + c, x);

                      2
    sqrt( - 4*a*c + b ) - b
{x=------------------------,
              2*a

                         2
     - (sqrt( - 4*a*c + b ) + b)
 x=-----------------------------}
                2*a
```

The standard solution is returned as a REDUCE list. We can change the order of the expression to get the usual form by using the korder declaration before calling solve:

```
17: korder b, a;
18: solve(a*x^2 + b*x + c, x);

          2
    sqrt(b  - 4*a*c) - b
{x=----------------------,
            2*a

              2
     - (sqrt(b  - 4*a*c) + b)
 x=--------------------------}
               2*a
```

If the solve operator could not find an explicit solution, it may still be able to find one in terms of an equation for the unknown. Such a solution is identified by specifying the subsidiary equation as the argument of the operator `root_of`. If there are several unknowns, each solution will be a list of equations for the unknowns. For example,

```
solve(x^7 - x^6 + x^2 = 1, x) ;
{x=root_of(x_**6  + x_ + 1,x_,tag_1),x=1}
```

Note that `redSolve` turns off the NAT switch as otherwise the results will not be parsed correctly. `redSolve` will ensure that NAT is off for the duration of its execution. Turning on the NAT switch using the `switches` parameter results in an error.

There are several switches related to `solve`, in particular `cramer`, `multiplicities`, `fullroots`, `trigform` and `varopt`. See the REDUCE manual section 7.17 for details of these and of the methods used for solving equations.

Value

`redSolve` returns an object of class "`redcas.solve`" which contains the following elements:

solutions a list of the solutions as returned by REDUCE. Each solution is a character vector.

rsolutions a list of the solutions, where possible converted to appropriate R objects. Because each element of a solution may be of a different type (real, complex, string), each solution is a list rather than a vector.

nsolutions numeric. The number of solutions.

rc numeric. The return code. 0 is success, 1 failure

root_of a complex vector identifying which solutions are presented in terms of the REDUCE `root_of` operator. A solution may expressed one or more unknowns in terms of the roots of another equation. In this case the other equation is enclosed in `root_of()`. This slot is a complex vector with the real part identifying the solution containing `root_of()` and the imaginary part the index of the unknown in that solution. This item is intended for use in a future release.

eqns the vector of equations passed to `redSolve`

unknowns the vector of unknowns passed to `redSolve`

switches the vector of switches passed to `redSolve`

The `redcas.solve` object may be printed using the `print` method.

Author(s)

martin gregory

See Also

`print` to print a `redcas.solve` object, REDUCE manual section 7.17 for details of the REDUCE `solve` operator and `redStart` for creating a REDUCE session.

Examples

```
## Open a CSL session:
s1 <- redStart()

## can only run code if session was successfully started
if (is.numeric(s1)) {
   sol <- redSolve(s1, eqns=c("x+3y=7", "y-x=1"), unknowns=c("x", "y"))
   print(sol)

   ## close session:
   redClose(s1)
}
```

`redSplitOut` *convert the vector returned by redExec into commands and command output.*

Description

REDUCE output contains both the commands issued and the output from the commands. Since the output is of interest, redSplitOut separates the input vector into a list containing vectors of output, commands and the input to `redSplitOut`.

Usage

```
redSplitOut(x)
```

Arguments

`x`	a character vector containing REDUCE output

Details

Although it is possible to use the REDUCE switch ECHO to prevent display of commands, this does not prevent display of the sequence numbers for the commands. `redSplitOut` can be called by `redExec` to separate the output from the commands. It can also be used after the call if the `split` option was not used or on an arbitrary output file from REDUCE, for example the full output which can be retrieved by `redClose`.

Value

A list containing three elements:

`out`	the command output
`cmd`	the commands
`raw`	the input to `redSplitOut`, i.e. the combined cmd and out vectors.

Author(s)

Martin Gregory

See Also

redExec

Examples

```
result <- c("5: ",
            "{x=-1}",
            "",
            "6: ",
            "b := {{2,12},{11,1}}",
            "",
            "7: ",
            "8: ",
            "                - m",
            "1a := ---------------------",
            "                 2       2",
            "       sqrt(v(3)  + v(2) )",
            "",
            "9: ",
            "",
            "10: ")
split <- redSplitOut(result)
writeLines(split$out)
```

---

redStart *function to start a REDUCE session*

Description

Starts a REDUCE session using a pipe connection sending the output (log) to a temporary file.

Usage

```
redStart(dialect = "csl", dirpath, options, echo=FALSE)
```

Arguments

dialect	character. The version of lisp with which the REDUCE executable was built. Allowed values are csl, the default, or psl.
dirpath	character. Path to a directory containing the REDUCE executables. Optional, see details below.
options	character. Options for the csl version. See details.
echo	boolean. Should the ECHO switch be turned on. Default is FALSE to match the REDUCE default.

Details

redcas finds the REDUCE executables in two different ways. The first is the explicit specification of the path using the dirpath argument passed to the redStart function. If this does not contain the REDUCE executable, redStart stops executing on the assumption that a specific version was intended.

The second method is used if dirpath is not specified. This determines the path at package load time by searching the following three locations in turn until the executable is found:

- the environment variable REDUCE_EXEC exists and contains the path to a directory containing an executable;
- the R option reduce_exec exists and contains the path to a directory containing the executable;
- the REDUCE executable is found in a directory named in the environment variable PATH.

This approach provides the flexibility to have a default version while being able to call different versions explicitly.

While it is possible to start a second REDUCE session without closing the first, garbage collection closes its connection. A future release will fix this issue.

redStart carries out the following actions:

- creates the REDUCE session using a pipe connection and sending the REDUCE output to a temporary file (in REDUCE terminology referred to as a log);
- registers the session, acquiring a session identifier
- calls redExec to load the REDUCE functions which are part of the package and to set the ECHO switch if requested.
- returns the session identifier.

Value

The session identifier which is an integer >0 if the session started successfully. If the session could not be started, redStart terminates using stop. If no REDUCE executable was found returns FALSE.

Author(s)

martin gregory

See Also

showSessions to list active sessions and redClose for closing a session.

Examples

```
## Open a CSL session:
id1 <- redStart()
## can only run code if session was successfully started
if (is.numeric(id1)) {
   ## show session details:
```

```
   print(showSessions())
   ## close session:
   redClose(id1)
}
```

---

showSessions *function to list details for currently active REDUCE sessions.*

Description

For each active REDUCE session returns the full path to the REDUCE executable and the temporary log file, the connection identifier, the number of code blocks executed and the number of lines read from the log so far.

Usage

```
showSessions(id)
```

Arguments

id	The session id for which to display the details. May be omitted. If none specified, display details for all sessions. The session id is returned by redStart.

Value

A data frame with the following columns:

id	integer, the session identifier;
pcon	integer, the input connection for the REDUCE session;
cmd	character, full path to the REDUCE executable;
logname	character, full path to the log file;
blockn	integer, number of blocks of REDUCE code submitted, i.e. number of calls to any function which sends code to REDUCE, currently redExec, asltx and redSolve ;
lines.read	integer, number of lines read from the log file for the session.

Author(s)

martin gregory

See Also

redStart for creating a REDUCE session and redClose for closing a session.

Examples

```
## Open a PSL session:
id1 <- redStart('psl')
## can only run code if session was successfully started
if (is.numeric(id1)) {
   ## show session details:
   print(showSessions(id1))

   ## retrieve the entire transcript up to now
   writeLines(readLines(showSessions(id1)[,'logname'], warn=FALSE))

   ## close session:
   redClose(id1)
}
```

Index

∗ **classes**
    redcas.solve, 9
∗ **methods**
    print, 7
∗ **package**
    redcas-package, 2

asltx, *2*, 3, *22*

list, *10*

print, 7, *9*, *10*, *18*
print,redcas.solve-method *(*print*)*, 7

redcas *(*redcas-package*)*, 2
redcas-package, 2
redcas.solve, *2*, *7*, 9, *18*
redcas.solve-class *(*redcas.solve*)*, 9
redClose, *2*, 10, *19*, *21*, *22*
redCodeDir, 11
redDropOut, *2*, 12, *15*
redExec, *2*, *4*, *6*, *11*, *13*, 13, *16*, *19–22*
redExpand, *2*, 15
redSolve, *2*, *7*, *9*, *10*, 16, *22*
redSplitOut, *2*, *15*, 19
redStart, *2*, *4*, *11*, *13*, *15*, *17*, *18*, 20, *22*

showSessions, *11*, *21*, 22

vector, *10*