BREDUCE User Manual

Thomas Sturm, FIM, Universität Passau, Germany*

January 7, 2008

Abstract

This document serves as a user guide for BREDUCE. This is a shell script plus a supporting REDUCE module which together admit batch processing of REDUCE jobs under Unix. BREDUCE takes as input several REDUCE files containing input data plus a configuration file. It produces as output a LaTeXdocument with the formatted results of all computations specified in the configuration file. Important features include the systematic application of all combinations of a choice of REDUCE switches and a mechanism for limiting the runtime of the single computations.

1 Introduction

[Hea04]

2 Installation of breduce

The BREDUCE distribution comprises three files:

1. An executable program `breduce`,
2. a supporting REDUCE module `breduce.red`,
3. this manual `breduce.tex`.

Make sure that `breduce` is executable, and put it into a directory of your choice, which you probably want to have within your search path. Put `breduce.red` into the same directory as `breduce`.

3 A Simple First Example

We are now going to describe an extremely simple example application of BREDUCE. This will help the users to verify that their installation is correct. Furthermore it will give good first idea about what BREDUCE does. Finally it will explain a major part of the files maintained by BREDUCE.

BREDUCE is based on the concept of a *series* of computations to be performed. A series consists of one or several *instances*. The user chooses a name

*sturm@redlog.eu

`<name>` for the series. The central file describing the series is the BREDUCE *configuration file* `<name>.breduce`.

For our first example here, all relevant information is contained in our configuration file `fac.breduce`. For convenience, we create a new directory `bredex1` to work in. Our `fac.breduce` reads as follows:

```
REDUCE='reduce'
seriesinstances='1 2 3 4 5 6 7 8 9 10'
command='factorize'
```

The reader might have to adapt the choice for `REDUCE`, which is the name of the executable REDUCE. The choice `reduce` above is also the default used if `REDUCE` is not specified at all. There are rather rigorous formal restrictions on the entries in configuration files:

- A *valid* line either starts with a BREDUCE *keyword*, such as the 3 lines above, or contains exclusively whitespace, or starts with the *comment sign* '`%`'.
- In keyword lines, there must not be any whitespace at the beginning or at the end or around '`=`'. The right hand side of '`=`' must be quoted with either single or double quotes.[1]

We now start our first BREDUCE jobs as follows:

```
breduce fac.breduce
```

where the extension is optional in the style of LaTeX.

Within a few milliseconds we obtain a file LaTeX `fac.tex` containing the table in Figure 4. In addition, `fac.tex` contains the name of the working directory (`bredex1` for our example) followed by a dump of `fac.breduce`. Since BREDUCE uses the LaTeX package "longtable," which allows have tables spread over several pages, it may be necessary to process `fac.tex` several times until the table is properly arranged. The same holds for this manual. We are going to discuss and considerably improve the content of the result column of the table in Figure 4 in the next section.

In addition to `fac.tex` BREDUCE creates a directory `fac/`, which contains log files `instance1-0.log`, ..., `instance10-0.log` of the 10 REDUCE runs. We are going to discuss the purpose of the appended '`-0`' later on in Section 5.

4 Processing Results

The result column in Figure certainly looks a bit disappointing: It contains the raw Lisp representations of the results, which have actually been obtained in the algebraic mode. Moreover, these Lisp representations are processed with LaTeX, which would cause problems whenever they happen to contain special LaTeX symbols. Finally, it is clear that the output of the full result would soon exceed the space available in a table when factorizing only slightly larger numbers.

[1] Technically, BREDUCE passes the keyword lines to `eval` in a Bash. Readers who do not understand what this means or feel they do not fully understand the possible consequences of this are strongly recommended to exclusively use single quotes! For advanced users the careful use of double quotes provides some hook to add information from the OS level.

sturm using reduce on lennier.local (i386 Darwin)

start of job on Tue Jan 1 10:32:03 CET 2008

instance	result	time (ms)
1	(list 1 1)	0
2	(list 2 1)	0
3	(list 3 1)	0
4	(list 2 2)	0
5	(list 5 1)	0
6	(list 2 1)(list 3 1)	0
7	(list 7 1)	0
8	(list 2 3)	0
9	(list 3 2)	0
10	(list 2 1)(list 5 1)	0

end of job on Tue Jan 1 10:32:03 CET 2008

Figure 1: `fac.tex`

To address all these issues, BREDUCE provides an optional keyword `processresults`. For instance

```
processresults='breduce_verbatim'
```

Puts the result into a LaTeX `verbatim` environment such that it may contain arbitrary characters. Another nice choice for small mathematical results is `breduce_tex`, which uses the `reduce` package `tri` [ASW89] for producing LaTeX-formatted output. For factorizing larger numbers in our examples the user might want to use `length` for obtaining the number of different factors.

Generally, with large results one can provide a self-written procedure for analyzing the results and outputting suitable LaTeX source code. The specification for such a procedure is as follows:

- It is called in the algebraic mode with the result of the application of the specified `command` as its only argument.
- It returns a either string, which is suitable to be processed by LaTeX, or an algebraic mode list of such strings. With the second variant all strings in the list are successively output.

We are going to see in Sections 6 and 10 where such self-written procedure can possibly be placed.

BREDUCE itself provides one such function: `breduce_processformula` analyzes results of `redlog` [DS97] quantifier elimination `rlqe`. If the result is "true" or "false," then this is printed. Otherwise, if all quantifiers have been successfully eliminated there is "$\top$" output, else "$\bot$." These symbols are followed by "$(\langle k\rangle \mathrm{q}/\langle n\rangle \mathrm{at})$" giving the numbers $\langle k\rangle$ of quantifiers and of atomic formulas $\langle n\rangle$ contained in the result.

5 Switches

The systematic evaluation of switches was one major motivation for the development of BREDUCE: With computer algebra software developers are often faced with design alternatives where it is not at all clear which choice is the best one. It is a helpful technique to introduce at least temporarily REDUCE switches which allow to interactively choose between the options. This allows to experiment with the options and to experimentally adjust the switches to default settings that deliver satisfactory performance and quality of results for most input data. At a later stage, there are usually some switches completely removed while others remain in order to allow to adapt the software to certain special input data which benefits from alternative settings.

The idea is now to use BREDUCE in this development process for systematically evaluating the effect of all possible combinations of certain switches.

For illustration, we use two switches. The switch `nopowers` is off by default. When on, multiple factors are collected in one flat list rather than building a pair from each factor and its multiplicity. The switch `rounded` toggles the use of fixed-precision floating point numbers, which does not have any relevant effect on the computation considered here but illustrates the treatment of multiple switches. The following `fac2.breduce` serves also as an example for the result processing option `breduce_tex` discussed in the previous section:

```
REDUCE='reduce'
seriesinstances='7 8 9 10'
switches='nopowers rounded'
command='factorize'
processresult='breduce_tex'
```

The table obtained in `fac2.tex` is displayed in Figure 2.

In the directory `fac2/` we obtain REDUCE log files named `instance1-0`, ..., `instance1-3`, `instance2-0`, ..., `instance5-3`. Notice that switch settings are arranged in such a way that they correspond to the binary expansion of the appended number "`-<n>`".

6 Packages and Initcommands

The keyword `packages` contains a whitespace-separated list of packages to be loaded. Here is an example:

```
packages='groebner groebnr2'
```

The keyword `initcommands` takes REDUCE commands, which are executed right after the packages are loaded, as for instance

```
initcommands='on gc,gsugar; in "my.red"; torder lex;'
```

Notice that here the switches are simply switched on in contrast to being systematically tested as with the `switch` keyword discussed in the previous section. It may in fact make sense to switch on `gc` in order to obtain corresponding information from the log files. The file `my.red` could provide suitable procedures for `processresult` as discussed in Section 4.

sturm using reduce on lennier.local (i386 Darwin)

start of job on Thu Jan 3 18:14:47 CET 2008

instance	nopowers	rounded	result	time (ms)
7	○	○	$\{\{7,1\}\}$	0
7	○	●	$\{\{7,1\}\}$	0
7	●	○	$\{7\}$	0
7	●	●	$\{7\}$	0
8	○	○	$\{\{2,3\}\}$	0
8	○	●	$\{\{2,3\}\}$	0
8	●	○	$\{2,2,2\}$	0
8	●	●	$\{2,2,2\}$	0
9	○	○	$\{\{3,2\}\}$	0
9	○	●	$\{\{3,2\}\}$	0
9	●	○	$\{3,3\}$	0
9	●	●	$\{3,3\}$	0
10	○	○	$\{\{2,1\},\{5,1\}\}$	0
10	○	●	$\{\{2,1\},\{5,1\}\}$	0
10	●	○	$\{2,5\}$	0
10	●	●	$\{2,5\}$	0

end of job on Thu Jan 3 18:14:49 CET 2008

Figure 2: `fac2.tex`

In contrast to all other keywords, the content of `initcommands` is not a whitespace-separated list. Instead it has to be specified in such a way that it can be literally pasted into REDUCE.

7 Commands

In this section we want to discuss in more detail the keyword `command`, which we had introduced already in Section 3.

To start with, we can specify a list of procedure names as for instance

```
commands='factorize factorial'
```

which is interpreted as a nested function call `factorize(factorial(...))`. The same specification holds for `processresult`. There is a subtle point about whether to include a procedure into `command` or into `processresult`: The time measurement given in the last column of the generated LaTeX include all procedures in `command` but not those in `processresult`. In order to obtain precise timings for the computations, the procedures in command are executed in REDUCE with a trailing '`$`' such that the timings do not include any printing times. Users explicitly interested in including printing times would add `write` as the leftmost procedure in `command`.

There is one apparent limitation of the approach: The rightmost procedure in `commands` and thus the entire chain of nested procedure calls specified by `commands` must establish a *unary* function. This can, however, easily be

worked around as follows: Imagine we actually want to work with a binary `myproc(x,y)`. Then we would provide an additional procedure

```
algebraic procedure myproc_nospread(l);
   myproc(first l,second l);
```

In the configuration file we would use this as follows:

```
commands='myproc myproc_nospread'
```

Consequently, for each instance of a series the two arguments for `myproc` have to be provided in two-element algebraic mode lists. This approach straightforwardly generalizes to arbitrary numbers of arguments. REDUCE provides `nth(<l>,<n>)` for obtaining the `<n>`-th element of a list `<l>`.

Recall the discussion on the systematic evaluation of switches at the beginning of Section 5. One would obviously like to similarly compare and evaluate alternative implementations `myfun1` and `myfun2` of some algorithm. For this one would introduce, say in a file `myfun.red`, corresponding switches and a procedure testing these switches:

```
switch use_myfun2;

algebraic procedure myfun0(x);
   if lisp !*use_myfun2 then myfun2 x else myfun1 x;
```

A suitable configuration file `myfun.breduce` would then contain

```
initcommands='in "myfun.red";'
switches='use_myfun2'
command='myfun0'
```

8 Random

Let us determine the number of (different) factors of some not too small random numbers. For this purpose we write the following configuration file `fac3a.breduce`:

```
switches='nopowers'
seriesinstances='random(10^37) random(10^38) random(10^39)'
command='factorize'
processresult='length'
```

The `reduce` procedure `random(<n>)` returns a random number between 0 and `<n>`. It is clear that for `off nopowers` the length of the list obtained from `factorize` is the number of different factors while for `on nopowers` we obtain the number of factors counting multiplicities. Since the output of `length` is simply a number we need not any `processresult` command. Looking at the obtained `fac3.tex` in Figure 3 we observe two things about our randomly generated numbers:

1. In each instance we obtain for both switch settings the same random number although these have been generated in different REDUCE sessions.

2. Comparing the random numbers of different sizes obtained in the various instances they turn out to be all very similar.

We are going to explain the situation without going into technical details: Since every single line in out table has been computed in a fresh `reduce` session, we have started every time with a random generator that has been freshly initialized in the very same way. To avoid identities between random numbers of the same size and the observed similarities for different sizes, we must explicitly initialize the random generator. BREDUCE offers four alternative styles of initialization. They can be specified via the keyword `random`:

(a) `random='breduce_instance'`

(b) `random='breduce_every'`

(c) `random='breduce_instance_abs'`

(d) `random='breduce_every_abs'`

`breduce_instance` uses the number of the current instance for initialization. Recall that instances are by definition the entries in `seriesinstances`. When trying all possible combinations of switches, these belong to the same instance, and would thus generate the same random numbers. This is a good choice for our example considered here. The result is displayed in Figure 4.

`breduce_every` initializes with the sum of the instance number and the decimal representation of the current switch setting. This results in a different initialization for every single row of the table as displayed in Figure 5. Note, however, that we are then not testing the switch combination for an instance on the same input data, which we probably want in most situations.

With both `breduce_instance` and `breduce_every` we still observe that there is the essentially same sequence of random numbers used every time. Since this might not really feel like random to some users there are `breduce_instance_all` and `breduce_every_all`. These are variants of the discussed options, which add for initialization also the wall clock time and date. Technically, it uses the number of seconds since the epoch in the sense of Unix (00:00:00 UTC, January 1, 1970).

In order to make also such BREDUCE jobs reproducible, the number of seconds used is dumped into the LaTeX output after the configuration file using the keyword `epoch`. When including this dumped `epoch` line into the configuration file, its value will be used instead of the real time and date and thus reproduce the result of the considered BREDUCE job.

9 Limiting the Computation Time

10 Instance Files

11 Signal Handling

12 Keyword Summary

`command`

`headline`

sturm using reduce on lennier.local (i386 Darwin)

start of job on Fri Jan 4 18:47:16 CET 2008

instance	nopowers	result	time (ms)
18708957911911717342849703520615247 75	∘	2	20
18708957911911717342849703520615247 75	•	3	20
91870895791191171734284970352061524775	∘	4	80
91870895791191171734284970352061524775	•	5	80
891870895791191171734284970352061524775	∘	6	20
891870895791191171734284970352061524775	•	7	20

end of job on Fri Jan 4 18:47:16 CET 2008

Figure 3: `fac3.tex`

sturm using reduce on lennier.local (i386 Darwin)
start of job on Sat Jan 5 09:30:14 CET 2008

instance	nopowers	result	time (ms)
18708957911911717342849703520615247 75	∘	2	20
18708957911911717342849703520615247 75	•	3	20
80834363874062020393350577869448966607	∘	2	870
80834363874062020393350577869448966607	•	2	870
46979783195693386905242618538683640843 9	∘	4	110
46979783195693386905242618538683640843 9	•	6	110

end of job on Sat Jan 5 09:30:18 CET 2008

Figure 4: `fac3a.tex`

sturm using reduce on lennier.local (i386 Darwin)

start of job on Sat Jan 5 09:33:09 CET 2008

instance	nopowers	result	time (ms)
18708957911911717342849703520615247 75	∘	2	20
83436387406202039335057786944896607	•	6	12080
80834363874062020393350577869448966607	∘	2	870
69797831956933869052426185386836408439	•	5	30
46979783195693386905242618538683640843 9	∘	4	110
25876130003980471771149179290422385027 1	•	5	5180

end of job on Sat Jan 5 09:33:28 CET 2008

Figure 5: `fac3b.tex`

`initcommands`

`killtime`

`packages`

`processresult`

`random`

`REDUCE`

`seriesinstances`

`seriesfilebasename`

`seriesprintname`

`switches`

References

[ASW89] Werner Antweiler, Andreas Strotmann, and Volker Winkelmann. A TEX-REDUCE-Interface. *ACM SIGSAM Bulletin*, 23(2):26–33, April 1989.

[DS97] Andreas Dolzmann and Thomas Sturm. Redlog: Computer algebra meets computer logic. *ACM SIGSAM Bulletin*, 31(2):2–9, June 1997.

[Hea04] Anthony C. Hearn. *Reduce User's Manual for Version 3.8*. Santa Monica, CA, February 2004. http://reduce-algebra.com/.