Bill J. Ellis
January 13, 2003

The MINI-CORPUS

Contents

1 README

```
########################################################################
#Mini-Corpus
#
#Bill J Ellis
#=======================================================================
#README
#-----------------------------------------------------------------------
#User manual for the Mini-Corpus.
########################################################################


About The Mini-Corpus
########################################################################
The Corpus provides an accessible store of SPARK problems.


Mini-Corpus Directory Structure
########################################################################
The Mini-Corpus exists as a large directory structure, based on the
following description:

Every example is a collection of code.  Often this can be compiled
to a binary. Different sections of this code can be analysed
in different ways to produce different verification problems.
For each verification problem there may be a number of solutions.

Thus there is a number of examples. Each example has a number of
corresponding problems. Each problem has a number of corresponding
solutions.

Hence the following structure:

|CORPUS
|--|examples
.  |--|EXAMPLE (1..a)
   .  |--|problems
      .  |--|PROBLEM (1..b)
         .  |--|solutions
            .  |--|SOLUTION (1..c)
               .  |
                  .
                  .
```

*Notes in this series are for ϵ-baked ideas, for $1 \geq \epsilon \geq 0$. Only exceptionally should they be cited or distributed outwith the NuSPADE project (EPSRC GR/R24081).

Each EXAMPLE, PROBLEM and SOLUTION has its own subdirectory, located in a directory of examples, problems and solutions respectively. The number of EXAMPLE, PROBLEM and SOLUTION directories can vary.

```
EXAMPLE
*************************************************************************

|EXAMPLE
|--|basedef  (Rules at example level)
   |source   (The source for this example)
   |problems (The problems for this example)
   |HTML     (Generated HTML)


PROBLEM
*************************************************************************

|PROBLEM
|--|extradef  (More rules at problem level)
   |genvcs    (How to generate VCs for this example)
   |--|patch  (Changes to the initial example source)
   |solutions (The solutions for this example)
   |HTML      (Generated HTML)


SOLUTION
*************************************************************************

|SOLUTION
|--|lemma     (Discovered rules at solution level)
   |info      (Status of the solution)
   |working   (Patches of example source)
   |--|source (A patched copy of the example source)
      |vcgs   (The EXAMINER generated vcgs)
   |HTML      (Generated HTML)
```

Installing The Mini-Corpus
###
The Mini-Corpus relies on some environment variables for shell based operation (browsing the generated HTML does not require any special variables).

These variables can be established in two ways:

As Required
****
Change directory into the root of the Mini-Corpus. Source the script CmdSouInitCorpus into your current shell environment:

```
source CmdSouInitCorpus
```

Hardwired
****
Extract from CmdSouInitCorpus the variable assignments required and include these in your shell profile.

It is assumed throughout this README that one of the above has been

```
performed.


Adding To The Corpus
#######################################################################
Given the nested structure it is necessary to add the details in
the order:
EXAMPLE, PROBLEM, SOLUTION


Normal Use
***********************************************************************
In most cases this involves adding some spark code, generating a
class of VCs and proving them.


Adding EXAMPLE
=======================================================================
Enter the corpus:
cd $LOC_DIR_CORPUS

Request addition:
CmdEvaAddToCorpus

Select 1: Add EXAMPLE to CORPUS:
1

Enter example name:
myexample

Go into the created example:
cd ./examples/myexample/

Add your SPARK code to the source directory:
cp -r MYSOURCE ./source

If you like, modify the provided Makefile to compile your program
(if possible):
vim ./source/Makefile


Adding PROBLEM (for EXAMPLE)
=======================================================================
Enter the corpus:
cd $LOC_DIR_CORPUS

Go into the example to add a problem to it:
cd ./examples/myexample/

Request addition:
CmdEvaAddToCorpus

Select 2: Add PROBLEM to EXAMPLE:
2

Enter problem name:
myproblem

Go into the created example:
```

```
cd ./problems/myproblem/

Customise ./genvcs:
vim ./genvcs/Makefile
vim ./genvcs/spark.sw
vim ./genvcs/target.dat
...


Adding SOLUTION (for PROBLEM)
=======================================================================
Enter the corpus:
cd $LOC_DIR_CORPUS

Go into the problem to add a solution to it:
cd ./examples/myexample/problems/myproblem/

Request addition:
CmdEvaAddToCorpus

Select 3: Add SOLUTION to PROBLEM:
3

Enter problem name:
mysolution

Go into the created solution:
cd ./solutions/myproblem/

If you like, change the status for this problem:
CmdEvaChangeStatus

Generate the VCs to be solved:
CmdEvaBuildWorking

Generate the HTML:
CmdEvaAllToHTML


Adding Rules
***********************************************************************
In the basedef, extradef or lemma directories rules can be added. The
correct directory depends on when the rule is expected to become
available.

Regular rule files (*.rul) are stored in the directory. So:
cp name.rul ./ruledir

Rule library files are stored in a subdirectory or extension (*.rpk)
(A rule package). So:
cd ruledir
mkdir name.rpk
cp MYRULLIBFILES ./name.rpk


A note on security
#######################################################################

Data:
```

```
As the data of the Corpus is held as a directory, it is as secure
as any regular directory. If the directory is on a web server, it
will be available to the entire world. If it is kept on a particular
machine then only users with login privileges and read access of
the particular directory will be able to access it.

Stability:
There is the potential for an error in the Corpus scripts to cause
some damage. To help overcome this, the scripts are filled with
checks to make sure things seem correct, and all rm operations
are confirmed with user interaction.

#######################################################################
#End of file
```

2 TECH

```
#######################################################################
#Mini-Corpus
#
#Bill J Ellis
#======================================================================
#TECH
#----------------------------------------------------------------------
#Some technical details about the Mini-Corpus.
#######################################################################


See The Source
#######################################################################
The source is fairly well documented and, given the unsophisticated
nature of the Mini-Corpus, is fairly straight forward to follow.
It is preferred to keep documentation as close to the corresponding
source as possible to reduce the chance of the two getting out
of sync.


Naming Files
#######################################################################
Conventions were developed for naming the various files. Each file
uses special identifiers to describe its mode of operation.


The first three letters:
***********************************************************************

Cmd
---
A top level command file. These are available for execution by
the user.

Sup
---
A support file. These should never be directly executed by the user.


The next three letters:
***********************************************************************
```

```
Eva
---
The file should be executed using 'eval'.  It is executed in its
own shell environment.

Sou
---
The file should be executed using 'source'.  It is executed inside
the current shell environment.

Fil
---
This is a not a file for execution. Its extension should determine
its purpose.


The remaining letters:
***********************************************************************

Gen...
------
This script contains code that generates a specific file. Higher
level scripts are not marked Gen, as they do not implement the
generation, rather they call generation scripts.

Gen...Info
----------
Generate an info file.

Gen...InfoToHTML
----------------
Generate HTML from an info file.


File extensions (file.extension):
***********************************************************************

gawk
----
A file for gawk, the GNU Projects implementation of the AWK
programming language.


Exceptions to the above:
***********************************************************************

about.txt
---------
Many scripts communicate via files. The format of these files are
described in 'about.txt' files.


The Role Of Info Files
#######################################################################
The process of generating HTML is separated into two steps:

+Generate an info file
+Convert this info file into HTML
```

By decomposing the problem in this manner the conversion to HTML is usually quite straight forward, with the majority of the effort being placed in generating the info file.

Each info file typically has an entry on each line which can be naturally converted into HTML. The nature of the lines and the names of the info files are described in the 'about.txt' files.

The Role Of CmdSouDoEnvironmentVariables

##

This script is at the centre of the Mini-Corpus. It is responsible for maintaining the physical location of every directory and file that may be accessed by another script. In doing so, changes to the structure of the Mini-Corpus will only require changes to this script.

Variable Conventions

##

The following conventions are observed for variable names.

LOC_DIR_ID

This is a directory path.
Is of the form: /dir/.../dir

LOC_DIR_ID_MORE

This is a directory path extension.
Is of the form: LOC_DIR_ID/more

LOC_FILE_ID_MORE_filename

This is a directory path and file.
Is of the form: LOC_DIR_ID_MORE/filename

NAME_ID

This is the name of a key directory.
Is of the form: dir

TEMP_ID

The path and file name of a temporary file.
Is of the form: /tmp/id

HTML_ID

The path and file name of a generated HTML file.
Is of the form: /dir/.../dir/file.html

INPUT_ID

A variable that gains its value from the parameters to a script. It should not be modified during execution of this script.

```
LOCAL_ID
--------
A variable that is not declared for export and intended for local
use in a script.  This careful definition is necessary as, where
scripts are sourced, their local variables accidentally fall into
the scope of the calling script.

SET_ID
------
A constant. These should only be set within
CmdSouDoEnvironmentVariables.


Error Handling
#######################################################################
Each script should set the special variable $? to a non
zero value if there is a error. After each script is called the
following test should always be made:

#Check exit status.
if [ "$?" -ne "0" ]; then exit "1"; fi

This ensures that any error propagates to the top level and the
entire program stops.


Debugging
#######################################################################
Inside CmdSouDoEnvironmentVariables there is the variable:
SET_DEBUG

Setting this to a value other than 0 switches on some additional
debugging information.

#######################################################################
#End of file
```