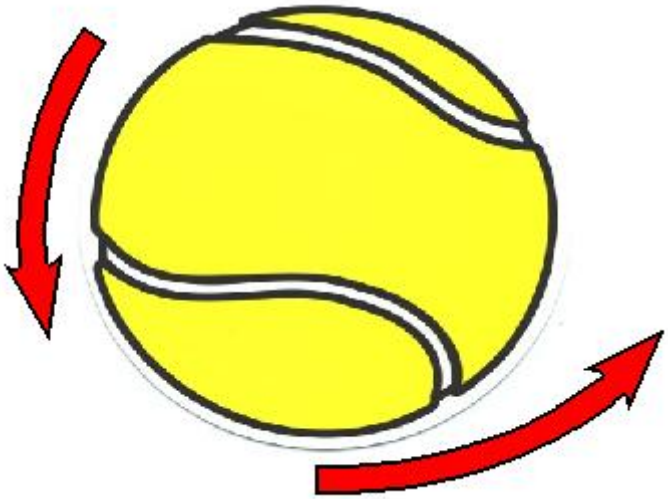
TopSpin

Version 2.2



Automatic symmetry reduction
for the SPIN model checker

User Manual

Alastair F. Donaldson

Before You Begin

What this manual covers

This user manual provides details of how to download, install and use TopSPIN, an automatic symmetry reduction tool for the SPIN model checker. TopSPIN can potentially aid in the verification of *safety* properties of concurrent systems specified in Promela.

- **Downloading and Installing** Chapter 1 provides details of other packages on which TopSPIN depends, including where these packages can be found, and explains how to download and install TopSPIN.
- **Worked Example** Chapter 2 provides a worked example showing how to run TopSPIN to obtain symmetry reduction for an example specification.
- **Overview of Options** An overview of TopSPIN options is presented in Chapter 3.
- **Limitations** A summary of limitations of TopSPIN is given in Chapter 4.
- **Compiling from Source** For users who wish to experiment further with TopSPIN, Chapter 5 explains how to obtain, compile and test the TopSPIN source code.
- **Troubleshooting and Bug Reporting** Chapter 6 presents solutions to common problems associated with the installation and operation of TopSPIN, and provides details of how bugs should be reported.

What this manual does not cover

- **Theory** Users interested in the theory on which TopSPIN is based should refer to relevant papers and a Ph.D. thesis, which are available from the tool web page (see below).
- **Use of SPIN** This manual assumes that the reader is familiar with the SPIN tool and the Promela language. Details of and documentation for the SPIN tool are available from the SPIN web page.[1]

Online resources

- **TopSPIN web page** `http://www.allydonaldson.co.uk/topspin/`
- **SourceForge** `https://www.sourceforge.net/projects/symmetryglasgow/`

1. `http://www.spinroot.com/`

Contents

1 Downloading and Installing

1.1 Prerequisites

TopSPIN is written in Java and GAP, interfaces with the GAP and SPIN packages, and produces C code which must then be compiled. Figure 1.1 summarises the packages which must be installed before TopSPIN can be used, and provides a URL for each package. For each package, the version used during the development of TopSPIN is specified. Use of these or newer versions is recommended.

Before going further, make sure each of the packages of Figure 1.1 is installed on your system. It is sufficient to download and install the GAP core package only. The archive of redistributed GAP packages, referred to as *packages-...* in the GAP installation instructions, is not mandatory.

Important Make sure that the SPIN tool is installed in such a way that it can be invoked by name from a command prompt, i.e. so that typing `spin` will launch SPIN. This is usually achieved by adding the folder for the SPIN executable to your *path* environment variable. To verify that SPIN is set up appropriately, type `spin` in a fresh command prompt. You should see something like:

```
C:\>spin
Spin Version 5.1.6 -- 9 May 2008
reading input from stdin:
```

If you have renamed the SPIN executable from `spin` to something like `spin-linux` or `spin516` then you will get errors when you run TopSPIN.

1.2 Downloading

To download TopSPIN version 2.2, go to the TopSPIN web page:

`http://www.allydonaldson.co.uk/topspin/`

and download the following file:

`TopSPIN_2.2.tgz`

Use the `tar` utility, the *WinRAR* tool,[1] or another suitable program to extract this archive to an appropriate location, e.g. `C:\Program Files\TopSPIN_2.2` under Windows, or `/usr/local/TopSPIN_2.2` under Linux. This location is referred to as the TopSPIN *root directory*.

The archive should contain the following:

- **`TopSPIN.jar`** Executable jar for the TopSPIN program
- **`documentation`** Folder containing this document, some related research papers and a Ph.D. thesis

1. `http://www.rarlab.com/`

Package	URL	Version
Java runtime environment	`http://java.sun.com/`	1.5.0_06
GAP system	`http://gap-system.org/`	4.4.6
SPIN model checker	`http://www.spinroot.com/`	5.1.7
GNU C Compiler (gcc)	`http://gcc.gnu.org/`	3.4.4

Figure 1.1: TopSPIN prerequisites.

- **`examples`** Folder containing example Promela specifications for use with TopSPIN
- **`Common`** Folder containing various GAP and C files files used by TopSPIN
- **`saucy`** Folder containing source code for the *saucy* program, which must be compiled (as described in §1.3.1) before TopSPIN can be used.

1.3 Installing

1.3.1 Compiling *saucy*

TopSPIN uses a prototype extension of the *saucy* program, which is used to compute symmetries of directed graphs. A version of *saucy* with this capability will eventually be available from the *saucy* website.[2] For the time being, a source distribution of *saucy* with the required extended functionality is provided with TopSPIN.[3]

Before using TopSPIN, you need to compile *saucy* on your platform. To do this, navigate to the `saucy` folder, which is inside the TopSPIN root directory, and type `make`:

```
C:\Program Files\TopSPIN_2.2>cd saucy
C:\Program Files\TopSPIN_2.2\saucy>make
gcc -ansi -pedantic -Wall -O3   -c -o main.o main.c
gcc -ansi -pedantic -Wall -O3   -c -o saucy.o saucy.c
gcc -ansi -pedantic -Wall -O3   -c -o saucyio.o saucyio.c
gcc  -o saucy main.o saucy.o saucyio.o
```

1.3.2 Creating a GAP workspace

In order to start GAP efficiently, TopSPIN requires a GAP *workspace* to be set up. Essentially, a workspace is an image of a GAP session with a selection of libraries and files already loaded and ready to be executed. For TopSPIN, the workspace consists of the GAP components of TopSPIN which have been developed for automatic symmetry reduction. These are in the `Common` folder, inside the TopSPIN root directory.

Navigate into the `Common` folder, and start GAP. GAP is usually started via a shell script (under Linux) or batch file (under Windows). If the folder containing this script/batch file is on your path then you should be able to start GAP simply by typing `gap`. Otherwise, start the tool by typing the full path for the script/batch file, e.g.

```
C:\gap4r4\bin\gap
```

or

```
/home/username/bin/gap
```

You should see something like this:

```
C:\Program Files\TopSPIN_2.2>cd Common

C:\Program Files\TopSPIN_2.2\Common>gap

C:\Program Files\TopSPIN_2.2\Common>rem sample batch file for GAP

C:\Program Files\TopSPIN_2.2\Common>C:\GAP4R4\bin\gapw95.exe -m 14m
-l C:\GAP4R4\

           #########           ######         ############           ###
         ##############        ######         #############        ####
        ###############        ########       #############       #####
       ################        ########       #####   ######      #####
      ######          #        #########      #####    #####     ######
     ######                   ##########      #####    #####    #######
     #####                    ##### ####      #####   ######   ########
     ####                    #####  #####     #############   ###  ####
     #####     #######       ####    ####     ############    ####  ####
```

2. `http://vlsicad.eecs.umich.edu/BK/SAUCY/`
3. Permission for including the *saucy* distribution with TopSPIN has been granted by Paul Darga, lead developer of *saucy*.

```
     #####     #######      #####    #####     ######        ####   ####
     #####     #######      #####    #####     #####         #############
      #####      #####     ################    #####         #############
      ######     #####     ################    #####         #############
      ################    ##################   #####                ####
       ###############    #####        #####   #####                ####
         #############    #####        #####   #####                ####
          #########      #####          #####  #####                ####

     Information at:  http://www.gap-system.org
     Try '?help' for help. See also  '?copyright' and  '?authors'

   Loading the library. Please be patient, this may take a while.
GAP4, Version: 4.4.6 of 02-Sep-2005, i686-pc-cygwin-gcc
Components:  ...
gap>
```

Now create a workspace as follows:

```
gap> Read("WorkspaceGenerator.gap");
gap> SaveWorkspace("gapworkspace");
true
gap> quit;
```

Ensure that these commands are typed *exactly* as shown, ensuring that the semi-colon is included after the `quit` command. When you exit GAP, you should find a file called `gapworksapce` in the `Common` directory.

1.4 Executing the TopSPIN jar file

It should now be possible to run the TopSPIN jar file on no input to display a list of TopSPIN options:

```
C:\>java -jar "C:\Program Files\TopSPIN_2.2\TopSPIN_2.2.jar"

Error: no input file specified.

To run TopSPIN on an input file:
    [command-line options] <inputfile>
For help on command-line or config file option:
    help <option>
For list of options:
    help
```

Successfully displaying this message confirms that the Java runtime environment is correctly installed on your system, and that the TopSPIN jar file you have obtained is in working order. If TopSPIN does *not* correctly display its options, proceed to Chapter 6 to try to deduce what is wrong, and to file a bug report if necessary.

In Chapter 2, instructions for running TopSPIN to perform symmetry reduction on a Promela specification are given.

2 Worked Example

In this chapter, the basic workings of TopSPIN are illustrated via a worked example.

2.1 Loadbalancer specification

The `examples` folder in the TopSPIN root directory contains a file, `loadbalancer.p`. This is a Promela specification for a *loadbalancer*, which forwards requests from a pool of clients to a pool of servers in a fair manner.

Components in the loadbalancer are a set of 2 *server* and 4 *client* processes with associated communication channels, and a *loadbalancer* process with a dedicated input channel. The *load* of a server is the number of messages queued on its input channel. Client processes send requests to the loadbalancer, and if some server's channel is not full, the loadbalancer forwards a request nondeterministically to one of the least-loaded server queues. Each request contains a reference to the input channel of its associated client process, and the server designated by the loadbalancer uses this channel to service the request.

2.2 Applying SPIN to the loadbalancer specification

Before applying TopSPIN to this example, it is worth checking that SPIN is in good working order by model checking the example without symmetry reduction.

Navigate to the `examples` directory, and run the following commands:

```
C:\Program Files\TopSPIN_2.2\examples>spin -a loadbalancer.p
C:\Program Files\TopSPIN_2.2\examples>gcc -o pan -O2 -DSAFETY pan.c
C:\Program Files\TopSPIN_2.2\examples>pan -m100000
```

SPIN should successfully verify that the model associated with the specification is deadlock-free, producing output similar to:

```
(Spin Version 5.1.6 -- 9 May 2008)
        + Partial Order Reduction

Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      +

State-vector 108 byte, depth reached 73413, errors: 0
   170903 states, stored
   413074 states, matched
   583977 transitions (= stored+matched)
        6 atomic steps
hash conflicts:     48755 (resolved)

   24.974       memory usage (Mbyte)

unreached in proctype client
        line 20, state 6, "-end-"
        (1 of 6 states)
unreached in proctype loadBalancer
        line 34, state 13, "-end-"
        (1 of 13 states)
unreached in proctype server
```

```
        line 43, state 7, "-end-"
        (1 of 7 states)
unreached in proctype :init:
        (0 of 9 states)

pan: elapsed time 0.818 seconds pan: rate 208927.87 states/second
```

2.3 Setting up a TopSPIN configuration file

You are almost ready to use TopSPIN for symmetry reduction! When you invoke TopSPIN on a Promela specification, the tool requires that a file called `config.txt` is available in the current directory. This file tells TopSPIN where to find the GAP and *saucy* programs, the location of some common source code, and the values of various runtime options. The file consists of a series of lines, each of which has the form:

attribute=*value*

Three attributes are required in every configuration file:

1. **`gap`** – the absolute path for the shell script or batch file required to launch GAP. The value for the `gap` attribute must be such that typing this value at the command prompt is all that is required to launch the GAP program.
2. **`saucy`** – the absolute path for the *saucy* executable. Again, the value for the `saucy` attribute must be exactly what you type to run *saucy* from the command line.
3. **`common`** – the absolute path to the `Common` directory in the TopSPIN root directory, followed by a (back- or forward-, depending on your operating system) slash.

Under Windows, `config.txt` might look like this:

```
gap=C:\gap4r4\bin\gap.bat
saucy=C:\Program Files\TopSPIN_2.2\saucy\saucy.exe
common=C:\Program Files\TopSPIN_2.2\Common\
```

whereas a possible Linux version of `config.txt` could be:

```
gap=/home/username/bin/gap
saucy=/users/grad/ally/TopSPIN_2.2/saucy/saucy
common=/users/grad/ally/TopSPIN_2.2/Common/
```

Note that the `gap` option should *not* specify the GAP executable itself, rather the shell script (`gap.sh`) or batch file (`gap.bat`) used to launch GAP.

The remainder of `config.txt` is used to specify TopSPIN options for a particular specification. One of these options is discussed in §2.5, and a complete overview of options is given in Chapter 3.

A configuration file must always be present in the directory from where you invoke TopSPIN. You will probably use different configuration files for different specifications, depending on the nature of the symmetry associated with these specifications. However, since the `gap`, `saucy` and `common` attributes are likely to be the same in all configuration files, it makes sense to keep a "skeleton" configuration file containing just these options, which you can then copy and extend for a given Promela specification.

2.4 Symmetry reduction with the *fast* strategy

When you run TopSPIN, you can specify a symmetry reduction *strategy* for the tool to use. The choice of strategy influences the speed of symmetry reduction and the factor of reduction obtained though the use of symmetry. Some strategies provide the maximum possible state-space reduction due to symmetry, at the expense of a slow state-space search. Other strategies are more lightweight, providing potentially sub-optimal reduction, but executing more quickly.

By default, TopSPIN uses the *fast* strategy. In a nutshell, when using this strategy TopSPIN attempts to work out an efficient symmetry reduction algorithm based on the type of symmetry associated with the input specification. The symmetry reduction algorithm used is *approximate* in the sense that it may result in exploration of more than one state per symmetric equivalence class.

2.4.1 Running TopSPIN

Copy `config.txt`, the basic configuration file created in §2.3, into the `examples` directory, and run TopSPIN on `loadbalancer.p` as follows:[1]

```
C:\Program Files\TopSPIN_2.2\examples>cp ../config.txt .
C:\Program Files\TopSPIN_2.2\examples>java -jar
          "C:\Program Files\TopSPIN_2.2\TopSPIN.jar" loadbalancer.p
File: loadbalancer.p
-------------------------------------
TopSPIN version 2.2
-------------------------------------
Configuration settings:
    Symmetry detection method: static channel diagram analysis
    Using 0 random conjugates
    Timeout for finding largest valid subgroup: 0 seconds
    Reduction strategy: FAST
    Using transpositions to represent permutations: true
    Using stabiliser chain for enumeration: true
    Using vectorisation: false
-------------------------------------

Typechecking input specification...

Specification is well typed!

Launching saucy via the following command: C:\Program Files\TopSPIN_2.2\saucy\saucy.exe -d
   "C:\Program Files\TopSPIN_2.2\Common\graph.saucy"

Starting GAP with command: C:\gap4r4\bin\gap.bat -L
   "C:\Program Files\TopSPIN_2.2\Common\gapworkspace" -q

The group:
   G = <(3 4)(clients2 clients1),(3 2)(clients0 clients1),(servers1 servers0)(7 6),
        (5 4)(clients2 clients3)>
is a valid group for symmetry reduction.

Generating symmetry reduction algorithms

The symmetry group has size 48
Completed generation of sympan verifier which includes algorithms for symmetry reduction!

To generate an executable verifier use the following command:
   gcc -o sympan sympan.c group.c
together with SPIN compile-time directives for your specification.

Execute the verifier using the following command:
   sympan.exe
together with SPIN run-time options for your specification.
```

If TopSPIN does *not* produce output like this, then proceed to Chapter 6 to try to deduce what is wrong, and to file a bug report if necessary.

The first part of the TopSPIN output specifies which version of the tool is being used, and how the various tool options have been configured. These options have all been set to default values. For now, simply note that the *fast* strategy is being used.

The tool then typechecks the input specification, reporting that the specification is well-typed.

After typechecking, the *saucy* and GAP programs are launched, to perform automatic symmetry detection. Note that the paths to these tools have been taken from the configuration file. The *saucy* tool is passed the `-d` option to indicate that the graph it will operate on is *directed*. A temporary file, `graph.saucy`, is also passed to the program: this is a graph generated by TopSPIN from which symmetries of `loadbalancer.p` are derived. The GAP package is launched with the `-L` option to indicate that a workspace (§1.3.2) should be loaded. The path to this workspace – `gapworkspace` inside the `Common` directory – is also passed to GAP. The `-q` operation launches GAP in *quiet* mode, which improves the efficiency of communication between the GAP and Java components of TopSPIN.

---

1. For reasons of space, some lines in the TopSPIN output have been wrapped. Also note that the output on your system will vary slightly according to the paths you have specified in `config.txt`.

TopSPIN successfully computes generators for a group of symmetries associated with the load-balancer specification. From the generators of this group, it can be seen that there is full symmetry between the *client* processes, as well as symmetry between the *server* processes. TopSPIN then goes on to generate symmetry reduction algorithms for this group, reporting that the number of symmetries is 48 (4! symmetries between the clients, and 2 symmetries between the servers, leads to $24 \times 2 = 48$ symmetries overall).

The end of the TopSPIN output details how the C files generated by SPIN and TopSPIN should be compiled and executed.

2.4.2 Compiling and executing the `sympan` verifier

The `examples` directory should now contain a number of files, including `sympan.c` and `group.c`. The `sympan.c` file is a modified version of the `pan.c` file produced by SPIN, which includes symmetry reduction algorithms. The `group.c` file includes functions for manipulating permutations.

Compile and execute this verifier using commands analogous to those shown in §2.2:

```
C:\Program Files\TopSPIN_2.2\examples>gcc -o sympan -O2 -DSAFETY sympan.c group.c
C:\Program Files\TopSPIN_2.2\examples>sympan -m100000

(Spin Version 5.1.6 -- 9 May 2008)
        + Partial Order Reduction

Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      +

State-vector 108 byte, depth reached 2323, errors: 0
     4960 states, stored
    12254 states, matched
    17214 transitions (= stored+matched)
        6 atomic steps
hash conflicts:        35 (resolved)

    5.735       memory usage (Mbyte)

unreached in proctype client
        line 20, state 6, "-end-"
        (1 of 6 states)
unreached in proctype loadBalancer
        line 34, state 13, "-end-"
        (1 of 13 states)
unreached in proctype server
        line 43, state 7, "-end-"
        (1 of 7 states)
unreached in proctype :init:
        (0 of 9 states)

pan: elapsed time 0.369 seconds pan: rate 13441.734 states/second
```

Comparing this model checking result with the result without symmetry reduction (§2.2) shows that the *fast* strategy leads to a state-space reduction factor of 34.5. The theoretical maximum symmetry reduction factor is 48, which is the size of the symmetry group computed by TopSPIN, so this is a reasonably good result. Notice that speedup factor is just 2.2, and as a result the *states/second* rate for verification is significantly lower when symmetry reduction is applied.

2.5 Symmetry reduction with the *enumerate* strategy

Although the *fast* strategy provides effective symmetry reduction for the loadbalancer example, the strategy does not provide space-optimal symmetry reduction. The *enumerate* strategy, on the other hand, is guaranteed to provide full symmetry reduction.

To use the *enumerate* strategy, open `config.txt` in the `examples` directory, and add the following line:

```
strategy=enumerate
```

This additional option tells TopSPIN to use the *enumerate* strategy over the default *fast* strategy.

Now apply TopSPIN to the loadbalancer specification, and compile and run the generated verifier:

```
C:\Program Files\TopSPIN_2.2\examples>java -jar
"C:\Program Files\TopSPIN_2.2\TopSPIN.jar" loadbalancer.p

   ... TopSPIN output ...

C:\Program Files\TopSPIN_2.2\examples>gcc -o sympan -O2 -DSAFETY sympan.c group.c
C:\Program Files\TopSPIN_2.2\examples>sympan -m100000

(Spin Version 5.1.6 -- 9 May 2008)
        + Partial Order Reduction

Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      +

State-vector 108 byte, depth reached 1916, errors: 0
     4213 states, stored
    11181 states, matched
    15394 transitions (= stored+matched)
        6 atomic steps
hash conflicts:        23 (resolved)

    5.638       memory usage (Mbyte)

unreached in proctype client
        line 20, state 6, "-end-"
        (1 of 6 states)
unreached in proctype loadBalancer
        line 34, state 13, "-end-"
        (1 of 13 states)
unreached in proctype server
        line 43, state 7, "-end-"
        (1 of 7 states)
unreached in proctype :init:
        (0 of 9 states)

pan: elapsed time 0.647 seconds pan: rate  6511.592 states/second
```

Verification using the *enumerate* strategy provides a better reduction factor: 40.6 vs. 34.5 with the *fast* strategy (§2.4). This comes at an expense: the speedup factor is reduced from 2.2 for the *fast* strategy to 1.3. For specifications with more symmetric components, this time penalty is more significant: the *enumerate* strategy works by applying *every* symmetry associated with the input specification to *every* state encountered during model checking. Since a specification with n symmetric components has a symmetry group of size at least $n!$, use of the *enumerate* strategy quickly becomes infeasible.

2.6 Summary so far

You should now have successfully installed TopSPIN and its prerequisite components, created a configuration file, and applied the tool to a Promela example. If you have encountered any problems during this process then proceed to Chapter 6 for troubleshooting ideas and information on how to report TopSPIN bugs.

At this stage, you may have learned all you need to know about TopSPIN for your basic symmetry reduction needs. Chapters 3 and 5 are for advanced users who wish to explore TopSPIN's more sophisticated options, and compile the tool from source, respectively.

3 Overview of Options

This chapter will be expanded into a complete reference for the various TopSPIN options. At the moment, every option is listed, but some of the descriptions are minimal.

3.1 Online help

Running TopSPIN with a single argument, `help`, displays a list of command-line switches and configuration file options for which short help messages can be generated.

Running TopSPIN with two arguments, `help <option>`, where `<option>` is one of the options for which help is available, displays a short message describing the function of the option.

3.2 Command-line options

3.2.1 `-check`

Apply this switch to just type-check a specification. Type checking will be symmetry-aware: the checker will report errors if the specification contains features which will cause symmetry detection to fail, *e.g.* arithmetic on `pid` variables, or absence of an `init` process.

When using the `-check` option it is not necessary to use `config.txt`, as the typechecking process does not depend on other tools or options.

3.2.2 `-detect`

Apply this switch to type-check and detect symmetry for a specification, but *not* apply symmetry reduction. TopSPIN will report the symmetries it computes, indicating whether there are problems in detecting symmetry. This option can be useful if you want to find out the extent to which symmetry is present in a model, but not actually do symmetry reduction.

Because this option does not require the output of symmetry reduction algorithms, TopSPIN will detect symmetries for specifications containing features which are not supported for symmetry reduction, *e.g.* a `never` claim. The idea here is that these features could, in principle, be supported fully, and determining whether symmetry is present may still be of interest.

3.2.3 `-relaxedarrayindexing`

TopSPIN is based on the ETCH typechecker, which tries to help modellers improve the quality of their specifications by performing relatively strict type-checking. In particular, ETCH does not allow an array to be indexed by an expression with a numeric type larger than `byte`. This is because the maximum size of an array in Promela is 255, which is also the largest `byte` value.

So, for example, the following code snippet will cause a type error:

```
mtype myArray[12];
int x;
x = ...;
myArray[x] = ...
```

due to the fact that `myArray` has been indexed by `x`, which has type `int`.

Sometimes, in practice, it is useful to index an array using a larger type than `byte`, *e.g.* so that a non-`byte` value like -1 can be used to represent a null index. For this reason, TopSPIN can be passed the

`-relaxedarrayindexing` command-line option, instructing the tool to allow indexing of arrays by expressions of *any* numeric type.

Note that the efficiency of a verification model can be improved by using smaller types: an `int` takes up more state-vector space than a `byte`, so the user should consider re-modelling their specification to use `byte` variables to index arrays where possible.

3.2.4 `-relaxedassignment`

As with array indexing, TopSPIN performs strict typechecking of numeric assignments by default, and rejects assignments from an expression of a "large" integer type (*e.g.* `short`) to a "smaller" integer type (*e.g.* `bit`). This is to prevent arithmetic overflow where possible.

So, for example, the following code snippet will cause a type error:

```
byte k;
k = -1;
```

due to the fact that `-1` has type `short`, which is a larger type than `byte`.

Because SPIN is not as strict as this, practical examples may rely on this kind of assignment. For this reason, specifying `-relaxedassignment` on the command line tells TopSPIN to allow a variable of any numeric type to be assigned the result of an expression of any numeric type.

As in §3.2.3, note that where possible, smaller numeric types should be used for reasons of efficiency. If you find you need the `-relaxedassignment` option then consider whether you could refactor your specification to use smaller types, removing the need for this option and reducing the size of the state-vector.

3.3 Mandatory configuration file options

3.3.1 `saucy`

String option specifying path to the *saucy* program. This option must be set by the user.

3.3.2 `common`

String option specifying path to the `Common` directory, which is part of the TopSPIN distribution. This option must be set by the user.

3.3.3 `gap`

String option specifying path to the GAP program. This option must be set by the user.

3.4 Configuration file options for symmetry detection

3.4.1 `explain`

TopSPIN detects symmetry by finding the *static channel diagram* for the input specification, computing its symmetries, and checking these symmetries for validity against the Promela specification. Some symmetries may not be valid: TopSPIN will try to compute the largest valid set of static channel diagram symmetries.

If you have written a specification which you expect to be symmetric, but for which TopSPIN reports invalid symmetries, it may be interesting to inspect the reason for this invalidity: it could indicate a typo in the specification, a misunderstanding on the part of the user, or a limitation of TopSPIN.

Setting `explain` to a positive integer n tells TopSPIN to output explanations for the first n invalid symmetries. An "explanation" for an invalid symmetry α consists of a pair of text files, one called m`_original.txt`, the other m`_permuted.txt` (where $m < n$). The *original* file shows the input specification before α is applied, then shows the specification after *normalization* has taken place. The *permuted* file shows the input specification after α has been applied, then shows the permuted

specification after normalization. TopSPIN regards a symmetry as valid if these normalized specifications are the same, so obviously they will not be identical for α.

The two text files can be compared with a diff tool (*e.g.* the excellent WinMerge[1]). Looking at the differences in the normalized parts should be instructive as to why the given symmetry is not valid.

By way of example, here is a mutual exclusion specification with five processes:

```
mtype = {N,T,C}
mtype st[6]=N

proctype user() {
  do
    :: d_step { st[_pid]==N -> st[_pid]=T }
    :: d_step { st[_pid]==T &&
         (st[1]!=C && st[2]!=C && st[1]!=C && st[4]!=C && st[5]!=C)
    -> st[_pid]=C
       }
    :: d_step { st[_pid]==C -> st[_pid]=N }
  od
}

init {
  atomic {
    run user();
    run user();
    run user();
    run user();
    run user();
  }
}
```

Running TopSPIN on this example, in verbose mode and with `explain=1`, produces output including the following:

```
Saucy computed the following generators for Aut(SCD(P)):
   Aut(SCD(P)) = <(3 2),(2 1),(3 4),(5 4)>
--------------------------------------
Computing valid generators:
    (3 2) : not valid
        generating explanation in files 0_original.txt and 0_permuted.txt
    (2 1) : not valid
    (3 4) : not valid
    (5 4) : valid
--------------------------------------
```

The tool has determined that the symmetry $(3\ 2)$ which swaps `user` processes 3 and 2 is not valid, and has written an explanation to the files `0_original.txt` and `0_permuted.txt`.

Figure 3.1 shows the results of comparing these files with WinMerge. This should alert the user to the fact that the operand `st[1] != C` appears twice in the highlighted boolean expression: this is clearly a mistake – one of these operands should be changed to `st[2] != C` to restore full symmetry and presumably correct the model.

This illustrates the idea that presence of symmetry can actually be a correctness requirement in its own right.

Default value: 0, *i.e.* no explanations will be given.

3.4.2 `timebound`

When detecting symmetry, TopSPIN may use a coset search to try to improve upon the initially computed group of symmetries. This search can be time-consuming on particular examples. The `timebound` integer option limits the length of time dedicated to this search to a maximum number of seconds.

Default value: 0, which is used to specify that the coset search should be unbounded.

3.4.3 `conjugates`

If the coset search is taking a long time, it is sometimes possible to speed things up by picking a number of candidate symmetries at random, finding the *conjugate* of each known valid generator by these

1. http://www.winmerge.org/

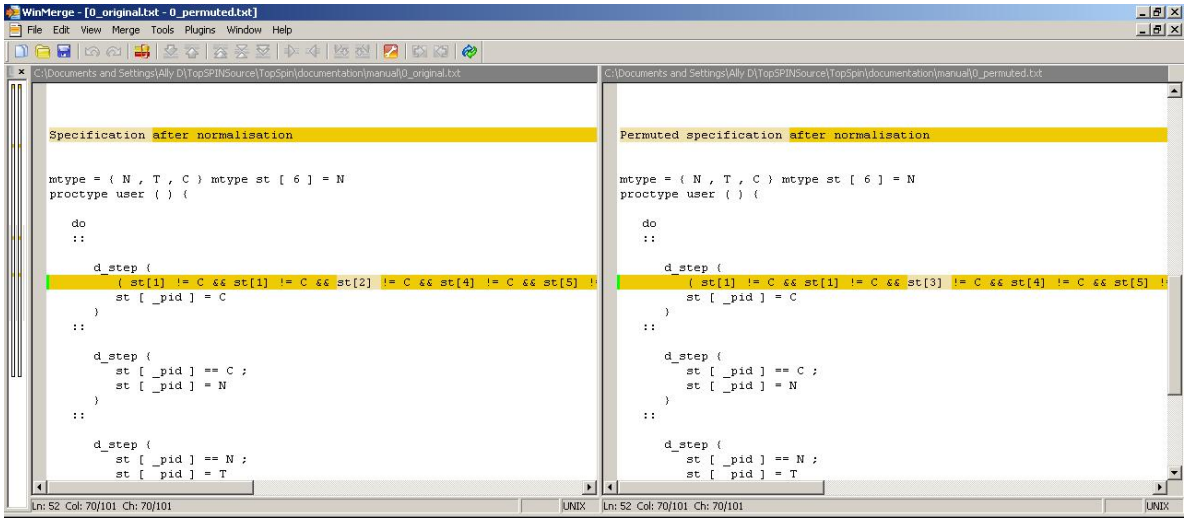


Figure 3.1: Comparing text files output by the `explain` option to see why symmetry $(3\ 2)$ is not valid.

random elements, and then testing the conjugates for validity. This is based on the practical observation that valid symmetries are often conjugate to other valid symmetries. The `conjugates` integer option is used to specify how many conjugates to use.
Default value: 0. If symmetry detection is slow, try 5 conjugates.

3.4.4 `symmetryfile`

Sometimes TopSPIN is unable to automatically detect symmetry from an input specification even though the user knows that symmetry exists. In this case, it is possible to specify a file containing generators for a symmetry group. The name of this file is specified via the `symmetryfile` option. If `symmetryfile` is set, TopSPIN will not attempt automatic symmetry detection, and will use the given group of symmetries for symmetry reduction.
Default value: no file specified, automatic symmetry detection will be attempted..

3.5 Configuration file options for symmetry reduction

3.5.1 `strategy`

String option specifying the strategy to use for symmetry reduction. Options are: fast, enumerate, hill-climbing, segment, flatten, exactmarkers, approxmarkers.
Default value: fast.

3.5.2 `transpositions`

Boolean option stating whether to apply group elements as transpositions.
Default value: true.

3.5.3 `stabiliserchain`

Boolean option stating whether to use a stabiliser chain for enumeration.
Default value: true.

3.5.4 `vectorise`

Boolean option stating whether to use vector SIMD instructions for symmetry reduction. The nature of the resulting vector code depends on the `target` configuration file option (see §3.5.7).
Default value: false.

3.5.5 `parallelise`

Boolean option stating whether to parallelise symmetry reduction using pthreads.
Default value: false.

3.5.6 `cores`

Integer option stating the number of cores available for parallel symmetry reduction.
Default value: 1.

3.5.7 `target`

String option specifying the target to use for vector and parallel symmetry reduction. Options are: PC, CELL, POWERPC.
Default value: not set by default.

3.6 Configuration file options for usability

3.6.1 `profile`

Profile the performance of TopSPIN?
Default value: false.

3.6.2 `verbose`

Display progress of TopSPIN in detail?
Default value: false.

3.6.3 `quiet`

Suppress non-vital output?
Default value: false.

4 Limitations

Before using TopSPIN for advanced verification it is important to understand what can and cannot be done with the tool. TopSPIN places various restrictions on the form a Promela specification may have. Also, certain SPIN options are not (yet) compatible with symmetry reduction.

This section briefly described the limitations of TopSPIN. Limitations marked with an asterisk could be removed with relative ease: they just haven't been dealt with yet. If you're working with SPIN and TopSPIN and are being hindered by these restrictions then please get in tough (see §6.4) and efforts will be made to add appropriate features to TopSPIN.

4.1 Process instantiation and dynamic process creation

TopSPIN requires all processes to be instantiated using `run` statements within an `init` process. The form of the `init` process must be:

```
init {
  atomic {
     run ...;
     run ...;
     ...
     run ...;
     <other statements>
  }
}
```

i.e. all process must be started atomically, and must come before any other statements in the `init` process (*e.g.* initialisation code), which must also appear within the `atomic` block.

In particular, TopSPIN does *not* support process instantiation using the `active` keyword, or via `run` statements outside the `init` process.

If a specification relies on dynamic process creation then it may be possible to re-model the processes as shown in Figures 4.1 and 4.2. Figure 4.1 shows a specification which instantiates copies of proctype P dynamically. Assuming that 3 is an upper bound for the number of instances of P which should be running at any time, Figure 4.2 shows an alternative way of expressing the specification. The proctype P now includes a channel parameter, and an instance of P waits until it can receive on this channel before executing its body. Its body is identical to the original, except that it includes a final `goto`

```
proctype P() {

  /* body */

}

proctype Q() {
   ...

   run P();

   ...
}
```

Figure 4.1: Skeleton Promela specification with dynamic process creation.

```
chan wakeup_P_1 = [0] of {bit};
chan wakeup_P_2 = [0] of {bit};
chan wakeup_P_3 = [0] of {bit}

proctype P(chan start) {

sleep:
   start?1;

   /* body */

   goto sleep
}

proctype Q() {
   ...

   if :: wakeup_P_1!1
      :: wakeup_P_2!1
      :: wakeup_P_3!1
   fi;

   ...
}

init {
   atomic {
      /* original 'run' statements */

      run P(wakeup_P_1);
      run P(wakeup_P_2);
      run P(wakeup_P_3)
   }
}
```

Figure 4.2: Re-modelled Promela specification without dynamic process creation.

statement after which it returns to its initial configuration.[1] The `init` process instantiates three copies of P, each with a distinct synchronous channel. Instead of instantiating a copy of P, the proctype Q now offers the literal value 1 to all channels on which instances of P may be listening. The example of Figures 4.1 and 4.2 can be adapted to handle multiple process types, with any fixed upper bound for each process type.

4.2 Process termination

Technically, process termination destroys symmetry in a Promela specification. SPIN only allows processes to terminate in *reverse* order. So, if five identical, terminating `client` proctypes are launched simultaneously with process identifiers in the range $\{1, 2, 3, 4, 5\}$, the processes will terminate one-by-one in a fixed order: 5, 4, 3, 2, 1. This ordering on process identifiers breaks symmetry between the processes.

Furthermore, the way symmetry reduction is implemented in TopSPIN is not strictly compatible with process termination. After the initial state, TopSPIN assumes a fixed set of running processes. If a process dies, TopSPIN may try to exchange this processes's state with that of another process when computing symmetry representatives, and this can (very occasionally) lead to verification-time errors (see §6.1.15).

If you want to work with terminating processes, you can insert a dummy termination state at the end of each proctype using a line like the following:

```
end_ok: false
```

1. This `goto` statement should really be part of an atomic block which also resets any local variables of the proctype to their initial values.

```
mtype = {N,T,C}
mtype st[2]=N

proctype user(byte id) {
  do
    :: d_step { st[id]==N -> st[id]=T }
    :: d_step { st[id]==T && st[0]!=C && st[1]!=C -> st[id]=C }
    :: d_step { st[id]==C -> st[id]=N }
  od
}

init {
  atomic {
    run user(0);
    run user(1);
  }
}
```

Figure 4.3: Promela specification which uses user-defined process identifiers.

```
mtype = {N,T,C} mtype st[3]=N;

proctype user() {
  do
    :: d_step { st[_pid]==N -> st[_pid]=T }
    :: d_step { st[_pid]==T && st[1]!=C && st[2]!=C -> st[_pid]=C }
    :: d_step { st[_pid]==C -> st[_pid]=N }
  od
}

init {
  atomic {
    run user();
    run user();
  }
}
```

Figure 4.4: Re-modelled specification which uses the `_pid` variable.

The `end` label ensures that SPIN will not complain about instances of the proctype blocking at this label, and the `false` statement ensures that proctype instances will never truly terminate.

4.3 The `_pid` variable should be used

For symmetry to be detected, it is important for proctypes to use their built-in `_pid` variable rather than a user-defined process identifier. This is illustrated in Figures 4.3 and 4.4. Processes in Figure 4.3 are parameterised by a *byte* identifier, which they use to index the `st` array. SymmExtractor is not yet sophisticated enough to work out the correspondence between the `id` parameter and the built-in identifier for each process. However, the specification can be converted into a form which SymmExtractor can handle, as shown in Figure 4.4. The disadvantage here is that position 0 of the array `st` is un-used, meaning that an array of size three rather than two is required, increasing the state-vector size by one byte. On the other hand, eliminating the `id` variables reduces the state-vector by two bytes, so the re-modelling works well for this example.

4.4 Restrictions on channels*

For simplicity, TopSPIN does not allow channel initialisers to be associated with channels which are declared locally to a proctype, and does not currently support arrays of channels.

4.5 Never claims, trace/notrace constructs, `accept` and `progress` labels

These are method for specifying liveness properties in a Promela model. TopSPIN is not yet compatible with verification of liveness properties, so you will get an error if you try to apply TopSPIN to a specification containing one of these constructs.

Symmetry *detection* is still possible with liveness verification constructs. It is OK to apply TopSPINto a specification containing a liveness construct with the `-detect` option (see §6.4). The tool will tell you any symmetries it detects; it's just not yet possible to exploit these symmetries when model checking.

4.6 Exclusive send/recieve (`xs`/`xr`) channel assertions

This is not actually a restriction, rather a warning for users wishing to use these keywords.

Promela includes keywords `xr` and `xs`, which stand for *exclusive receive* and *exclusive send* respectively. A process can include a declaration '`xr` *name*', where '*name*' is the name of a previously declared channel, to indicate that only this process can receive messages on the channel. The `xs` keyword is used similarly. Providing SPIN with this information can lead to more efficient partial-order reduction. It is not possible to check, statically, whether `xs` and `xr` are used correctly, but incorrect uses are flagged by SPIN during verification. These keywords do not affect the presence of symmetry in the model associated with a specification, so are no problem for symmetry detection.

However, there is a potential problem with exploiting `xs`/`xr` information in conjunction with symmetry reduction, described in Appendix C.1.1 of Donaldson's thesis. In practice, the problem with these keywords is unlikely to cause problems: there is a slim chance that if the keywords are used incorrectly, symmetry reduction will prevent this from being flagged up during verification. Therefore, TopSPIN *does* allow these features to be used, but it is the user's responsibility to use them with care.

4.7 Unsigned data type*

Promela supports an `unsigned` numeric type. A declaration of the form `unsigned` $x : y$ declares an integer variable x which takes non-negative values which can be represented using y bits. Clearly the use of this data type will have no effect on our symmetry detection/reduction techniques. However, TopSPIN is integrated with an enhanced Promela type checker which does not currently support the `unsigned` data type. A temporary fix for this omission is to replace each occurrence of the `unsigned` keyword with one of the other numeric types during symmetry detection.

4.8 Sorted send, random receive (`!!` and `??` operators)

Promela provides alternative channel operators: `!!` (sorted send) and `??` (random receive). Sending data on a buffered channel using `!!` causes messages to be queued on the channel in sorted order. Messages can be retrieved from the buffer in a random order using `??`. These operators provide a useful alternative to FIFO channel semantics. They also aid state-space reduction: storing channel contents in a sorted manner can be seen as a form of state canonicalisation. However, storing `pid` messages in a sorted queue imposes an ordering on the set of process identifiers. It is not immediately clear whether this ordering has an effect on symmetry.

For the time being, TopSPIN allows the `!!` and `??` operators to be used, but they should be used with care. If you get weird symmetry reduction results using these operators then please get in touch (§6.4).

4.9 Embedded C code

Recent versions of SPIN allow C code to be embedded in a Promela specification, and certain variables from the C part of the specification to be included in the SPIN state-vector. Automatic symmetry detection for this mix of C and Promela is beyond the scope of TopSPIN at present.

4.10 Breadth-first search*

TopSPIN has not been tested with breadth-first search, and will give an error message if `sympan.c` is compiled with `-DBFS`.

5 Compiling from Source

5.1 Downloading TopSPIN source code

The TopSPIN source code is stored in a Subversion repository, hosted by the Department of Computing Science at the University of Glasgow at the following URL:

```
https://ouen.dcs.gla.ac.uk/repos/symmetry/
```

Use subversion to check out the TopSPIN source code to an appropriate location:

```
C:\prog>svn checkout https://ouen.dcs.gla.ac.uk/repos/symmetry/trunk TopSPINSource
A TopSPINSource/TestModels
A TopSPINSource/TestModels/EtchTesting
A TopSPINSource/TestModels/EtchTesting/ParsePassTests
A TopSPINSource/TestModels/EtchTesting/ParsePassTests/peterson
A TopSPINSource/TestModels/EtchTesting/ParsePassTests/hello
A TopSPINSource/TestModels/EtchTesting/ParsePassTests/pathfinder
A TopSPINSource/TestModels/EtchTesting/ParsePassTests/snoopy
A TopSPINSource/TestModels/EtchTesting/ParsePassTests/mobile1
A TopSPINSource/TestModels/EtchTesting/ParsePassTests/mobile2
A TopSPINSource/TestModels/EtchTesting/ParsePassTests/pftp
A TopSPINSource/TestModels/EtchTesting/ParsePassTests/leader
A TopSPINSource/TestModels/EtchTesting/ParsePassTests/loops
...
A TopSPINSource/Common/Minimising.gap
A TopSPINSource/Common/parallel_symmetry_cell_ppu.c
A TopSPINSource/Common/Verify.gap
A TopSPINSource/Common/WorkspaceGenerator.gap
A TopSPINSource/Common/parallel_symmetry_cell_spu.c
A TopSPINSource/Common/parallel_symmetry_cell_ppu.h
A TopSPINSource/symmextractor_common_config.txt
Checked out revision 75.
```

5.2 Generating the Promela parser

TopSPIN is based on a Promela parser which is constructed using the SableCC parser generator. Download SableCC version 3.2 from the Sable website[1], and generate the parser as follows (adapting the command according to where you have installed SableCC):

```
java -jar C:\sablecc-3.2\lib\sablecc.jar promela.grammar
```

This command should result in output similar to the following:

```
C:\prog\TopSPINSource>java -jar C:\sablecc-3.2\lib\sablecc.jar
promela.grammar

SableCC version 3.2 Copyright (C) 1997-2003 Etienne M. Gagnon <etienne.gagnon@uqam.ca>
and others.  All rights reserved.

This software comes with ABSOLUTELY NO WARRANTY.  This is free software,
and you are welcome to redistribute it under certain conditions.

Type 'sablecc -license' to view the complete copyright notice and license.

 -- Generating parser for promela.grammar in C:\prog\TopSPINSource
Adding productions and alternative of section AST.
Verifying identifiers.
```

1. http://sablecc.org/

```
Verifying ast identifiers.
Adding empty productions and empty alternative transformation if necessary.
Adding productions and alternative transformation if necessary.
computing alternative symbol table identifiers.
Verifying production transform identifiers.
Verifying ast alternatives transform identifiers.
Generating token classes.
Generating production classes.
Generating alternative classes.
Generating analysis classes.
Generating utility classes.
Generating the lexer.
 State: INITIAL
 - Constructing NFA.
.....................................................................
.....................................................................
 - Constructing DFA.
.....................................................................
.....................................................................
 - resolving ACCEPT states.
Generating the parser.
.....................................................................
.....................................................................
```

Due to a bug in SableCC, it is then necessary to apply a patch file to the generated parser. Do this by executing the following command (you will need to have PERL installed: if you don't, you could easily re-write the PERL script in your favourite language!).

```
C:\prog\TopSPINSource>perl apply_patch.pl
```

5.3 Compiling and creating a jar

You have now downloaded and generated all the Java source code for TopSPIN, and can proceed to compile this source code. The `javac` compiler and `jar` utility are required to compile the source and create an executable jar file respectively. The source code is all contained in directories within `src`.

5.3.1 Compiling

There are two options for compilation:

1. Create an Eclipse project from the TopSPIN source code, in which case Eclipse will automatically compile the code.
2. Use the `make` utility and the supplied `Makefile` to compile the source code, by typing the command `make classes` in the TopSPIN root directory. The provided `Makefile` uses the `sed` and `find` utilities, which are provided as standard with Linux, and are available on Windows via Cygwin.

Compiling the source code using the `Makefile` gives output along the following lines:

```
C:\prog\TopSPINSource>make classes
javac src/etch/checker/Check.java
Note: .\src\promela\parser\Parser.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
javac src/etch/checker/CheckerTest.java
javac src/etch/testing/EtchTestCase.java
javac src/etch/testing/EtchTester.java
javac src/group/Group.java
javac src/promela/analysis/ReversedDepthFirstAdapter.java
javac src/symmextractor/InlineReplacer.java
javac src/symmextractor/SymmExtractor.java
javac src/symmextractor/testing/SymmExtractorFailTestOutcome.java
javac src/symmextractor/testing/SymmExtractorRunTestOutcome.java
javac src/symmextractor/testing/SymmExtractorTestCase.java
javac src/symmextractor/testing/SymmExtractorTester.java
javac src/symmreducer/testing/ModelCheckingResult.java
javac src/symmreducer/testing/SymmReducerFailTestOutcome.java
javac src/symmreducer/testing/SymmReducerTestCase.java
javac src/symmreducer/testing/SymmReducerTester.java
javac src/testing/RunAllTests.java
```

Note the compile warnings generated for `Parser.java`: these are due to code generated by SableCC. All other files should compile without warnings.

5.3.2 Creating a jar file

If you are using Eclipse then you can create an executable jar file for TopSPIN by exporting your TopSPIN project as a jar, and selecting `src.TopSpin` as the *main* class.

If using the supplied `Makefile`, typing `make jars` will produce two jar files:

```
C:\prog\TopSPINSource>make jars

jar cmf manifest.txt TopSPIN.jar src/etch/checker/Check.class src/etch/checker/Checker.class
... <many .class files> ...
src/utilities/Strategy.class src/utilities/StringHelper.class src/promela/lexer/lexer.dat
src/promela/parser/parser.dat && echo "TopSPIN.jar built successfully."
TopSPIN.jar built successfully.

jar cmf tests_manifest.txt TopSPINTests.jar src/etch/checker/Check.class
src/etch/checker/Checker.class
... <many .class files> ...
src/utilities/Strategy.class src/utilities/StringHelper.class src/promela/lexer/lexer.dat
src/promela/parser/parser.dat && echo "TopSPINTests.jar built successfully."
TopSPINTests.jar built successfully.
```

Note that you can skip the `make classes` step, and type `make jars` to both compile the Java files and produce jar files. To delete all jar and class files, use `make clean`.

5.4 Try your compiled version on an example

Now that you have successfully compiled a jar file from the TopSPIN source, you are effectively in the same position as a user who has downloaded the TopSPIN jar file using the instructions given in §1.2. Of course, you have the advantage of being able to modify TopSPIN to suit your purposes!

The next steps involve compiling *saucy*, creating a GAP workspace, and testing TopSPIN on a simple example. Therefore you should work your way through Chapters 1 and 2, using your compiled jar file in place of the downloaded jar file.

5.5 Acceptance Tests

The TopSPIN source checkout includes a reasonably large set of acceptance tests. It is highly recommended that you run these tests using the instructions below before commencing any development work on TopSPIN – passing the acceptance tests confirms that you are starting from a stable base. The acceptance tests can also be run regularly during development, to ensure that the addition of new features to TopSPIN does not adversely affect the tool's existing features.

Important Before running the acceptance tests, make sure that the GCC tool is installed in such a way that it can be invoked by name from a command prompt, i.e. so that typing `gcc` will launch GCC. To check this, try typing `gcc` from a fresh prompt. You should see something like:

```
C:\>gcc
gcc: no input files
```

5.5.1 Setting up a configuration file for testing

The TopSPIN test suite uses a called `symmextractor_common_config.txt` for some configuration options which apply to most tests. Open this file, and edit the `gap` line to use the appropriate command for your setup. You should not have to change the `Common` or `saucy` lines: the source code checkout is organised so that the `saucy` and `Common` directories are sub-directories of the directory in which `symmextractor_common_config.txt` is contained.

5.5.2 Running the tests

You can now run the acceptance tests via the `TopSPINTests.jar` file:

```
java -ea -jar TopSPINTests.jar 2> temp.txt
```

The `-ea` argument switches on assertion checking, which is useful for finding potential problems with TopSPIN. The final part of the command, `2> temp.txt`, pipes error messages generated during testing to the file `temp.txt`. This is useful since many of the tests are *fail* tests, which check that the TopSPIN typechecker correctly rejects Promela specifications which are not suitable for processing by TopSPIN. Redirecting these error messages to a text file means that the screen is not cluttered with error messages.

You should see something like this when you run the tests:

```
ETCH TESTS
==========
   [PASS] expected and actual: BadlyTyped, file: TestModels/EtchTesting/FailTests/failrec...
   [PASS] expected and actual: WellTyped, file: TestModels/EtchTesting/PassTests/testtele...
   [PASS] expected and actual: WellTyped, file: TestModels/EtchTesting/PassTests/test_ex_...
   [PASS] expected and actual: WellTyped, file: TestModels/EtchTesting/PassTests/testtele...
   [PASS] expected and actual: BadlyTyped, file: TestModels/EtchTesting/FailTests/faildup...
   [PASS] expected and actual: ParsePass, file: TestModels/EtchTesting/ParsePassTests/pet...
   ...


SYMMEXTRACTOR TESTS
===================
   [PASS] expected and actual: (well typed, group size = 72, coset search: no), file: Tes...
   [PASS] expected and actual: (well typed, group size = 1296, coset search: no), file: T...
   [PASS] expected and actual: BreaksRestrictions, file: TestModels/SymmExtractorTests/Ba...
   [PASS] expected and actual: (well typed, group size = 3628800, coset search: no), file...
   [PASS] expected and actual: BreaksRestrictions, file: TestModels/SymmExtractorTests/Ba...
   ...


SYMMREDUCER TESTS
=================
   [PASS] expected and actual: ((well typed, group size = 120, coset search: no), no. sta...
   [PASS] expected and actual: ((well typed, group size = 3628800, coset search: no), no....
   [PASS] expected and actual: ((well typed, group size = 6, coset search: no), no. state...
   [PASS] expected and actual: ((well typed, group size = 5040, coset search: no), no. st...
   [PASS] expected and actual: ((well typed, group size = 720, coset search: no), no. sta...
   ...


Summary:
   238 passes
   0 fails


Acceptance tests passed - you may commit your changes!
```

There are in the order of 250 test cases. These are divided into ETCH tests, which test the typechecking component of TopSPIN; SymmExtractor tests, which check the symmetry detection capabilities of the tool, and SymmReducer tests, which perform symmetry reduction on Promela examples, checking that verification results for these examples are as expected. The ETCH tests run very quickly, the SymmExtractor tests run at a moderate speed, and the SymmReducer tests run fairly slowly. Testing takes approximately 10 minutes on an average PC.

5.5.3 Problems running the tests

If you have compiled a fresh checkout of TopSPIN then all of the acceptance tests should pass, since their passing is a condition for committing changes to the source code repository. Therefore, if you have any problems running the tests this is likely due to a problem with the way you have set up GAP, *saucy*, SPIN, or GCC. Have a look at §6.1 which details common problems encountered when using TopSPIN. If test cases continue to fail then please send a bug report to the TopSPIN development team: see §6.2 for details.

6 Troubleshooting and Bug Reporting

This section includes a list of common problems with the use of TopSpin. Please also refer to current limitations of TopSpin, in Chapter 4.

6.1 Common problems

6.1.1 Missing configuration file

Problem: There is no file named `config.txt` in your working directory when you run TopSpin.

Example error message:

```
Error opening configuration file "config.txt", which should be
located in the directory from which you run TopSPIN.
```

Solution: Follow the instructions in §2.3 on how to create `config.txt`.

6.1.2 Incomplete configuration file

Problem: The file `config.txt` does not contain entries for all of the mandatory options, `gap`, `saucy` and `common`.

Example error message:

```
No configuration specified for GAP.
No configuration specified for saucy.
No common directory specified.
```

Solution: Follow the instructions in §2.3 on how to create `config.txt` with these mandatory options.

6.1.3 The *saucy* program is not correctly installed

Problem: You may not have correctly installed and compiled *saucy*, the graph automorphism program on which TopSpin relies.

Example error message:

```
Error launching saucy with command: C:\Program Files\TopSPIN\saucy\saucy.exe -d
"C:\Program Files\TopSPIN\Common\graph.saucy"
java.io.IOException: CreateProcess: C:\Program Files\TopSPIN\saucy\saucy.exe -d
"C:\Program Files\TopSPIN\Common\graph.saucy" error=2
        at java.lang.ProcessImpl.create(Native Method)
        ... rest of Java stack trace
```

Solution: Source code for *saucy* is included with the TopSpin distribution, which can be downloaded via the instructions of §1.2. However, you need to compile the *saucy* source code using GCC. To do this, follow the instructions given in §1.3.1.

6.1.4 Path to *saucy* in configuration file is wrong

Problem: From TopSpin's point of view this is the same as the previous problem. From your point of view there is a difference: you may have correctly installed and compiled *saucy*, but but mistyped the path to *saucy* in `config.txt`.

Example error message:

```
Error launching saucy with command: C:\Program Files\TopSPIN\saucy\tsaucy.exe -d
"C:\Program Files\TopSPIN\Common\graph.saucy"
java.io.IOException: CreateProcess: C:\Program Files\TopSPIN\saucy\tsaucy.exe -d
"C:\Program Files\TopSPIN\Common\graph.saucy" error=2
        at java.lang.ProcessImpl.create(Native Method)
        ... rest of Java stack trace
```

Note that TopSPIN is trying to launch `tsaucy.exe` rather than `saucy.exe`, due to a typo in `config.txt`.
Solution: Make sure *saucy* is correctly downloaded and compiled (§1.3.1), and ensure that `config.txt` contains a line for *saucy* with *exactly* the absolute path to the tool (§2.3).

6.1.5 GAP is not correctly installed

Problem: You may not have correctly installed GAP, the computational group theory package on which TopSPIN relies.

Example error message:

```
Starting GAP with command: C:\gap4r4\bin\gap.bat -L
"C:\Program Files\TopSPIN\Common\gapworkspace" -q
Exception in thread "main"
java.io.IOException: CreateProcess: C:\gap4r4\bin\gap.bat -L
"C:\Program Files\TopSPIN\Common\gapworkspace" -q error=2
        at java.lang.ProcessImpl.create(Native Method)
        ... rest of Java stack trace
```

Solution: Download and install GAP from the GAP website. The URL for this website and the version of GAP which TopSPIN supports are given in Figure 1.1 (§1.1).

6.1.6 Path to GAP in configuration file is wrong

Problem: From TopSPIN's point of view this is the same as the previous problem. From your point of view there is a difference: you may have correctly installed GAP, but mistyped the path to GAP in `config.txt`.

Example error message:

```
Starting GAP with command: C:\gap4r4\bin\tgap.bat -L
"C:\Program Files\TopSPIN\Common\gapworkspace" -q
Exception in thread "main"
java.io.IOException: CreateProcess: C:\gap4r4\bin\tgap.bat -L
"C:\Program Files\TopSPIN\Common\gapworkspace" -q error=2
        at java.lang.ProcessImpl.create(Native Method)
        ... rest of Java stack trace
```

Note that TopSPIN is trying to launch `tgap.bat` rather than `gap.bat`, due to a typo in `config.txt`.
Solution: Make sure you have correctly downloaded and installed GAP(§1.1), and ensure that `config.txt` contains a line for GAP with *exactly* the absolute path to the tool (§2.3).

6.1.7 Common directory does not exist, or user does not have permissions for this directory

Problem: The location of the TopSPIN `Common` directory has been specified incorrectly in `commmon.txt`.

Example error message:

```
Error while trying to create file "C:\Program Files\TopSPIN\Common\graph.saucy".
Make sure that the directory C:\Program Files\TopSPIN\Common\ exists,
and that you have write permission.
```

Solution: Make sure that `config.txt` contains a line of the form `common=`*name*, where *name* is the *absolute* path to the `Common` directory provided with the TopSPIN distribution. Make sure this path has not been mistyped, and that the path includes a terminating slash.

6.1.8 Path to Common directory does not have terminating slash

Problem: The location of the TopSPIN `Common` directory has been specified without a final forward- (Linux) or back- (Windows) slash.

Example error message:

Solution: Make sure that `config.txt` contains a line of the form `common=`*name*, where *name* is the *absolute* path to the `Common` directory provided with the TopSPIN distribution. Make sure this path has not been mistyped, and that the path includes a terminating slash.

6.1.9 Missing or corrupt GAP workspace

Problem: The file `gapworkspace` in the `Common` directory is either missing or corrupted.

Example error message:

```
Starting GAP with command: C:\gap4r4\bin\gap.bat -L
"C:\Program Files\TopSPIN\Common\gapworkspace" -q
Error -- bad GAP workspace specified in configuration file.

GAP produced errors:
====================
gap: Press <Enter> to end program
End of GAP errors
```

Solution: Check that:

1. You have followed the instructions in §1.3.2 on creating a GAP workspace
2. This workspace has been successfully saved in the `Common` directory
3. You have not changed the version of GAP you are using since creating the workspace
4. The workspace was created exactly the same machine, with the same operating system, on which you are running TopSPIN.

6.1.10 GAP executable specified in configuration file, instead of shell script/batch file

Problem: You have specified the path to the GAP executable as the `gap` option in `config.txt`.

Example error message:

```
GAP produced errors:
====================
Error, the library file 'system.g' must exist and be readable called from
CallFuncList( func, arg ); called from
RereadLib( "system.g" ); called from
<function>( <arguments> ) called from read-eval-loop
Entering break read-eval-print loop ...
you can 'quit;' to quit to outer loop, or
you can 'return;' to continue
brk>
```

Solution: Open `config.txt` and change the `gap` option to refer to the shell script (`gap.sh`) or batch file (`gap.bat`).

6.1.11 SPIN is not correctly installed

Problem: You have not correctly installed SPIN in such a way that the tool can be invoked by simply typing `spin`, as discussed in §1.1.

Example error message:

```
An error occurred while constructing the "sympan" files.
java.io.IOException: CreateProcess: spin -a loadbalancer.p error=2
        at java.lang.ProcessImpl.create(Native Method)
        ... rest of Java stack trace
```

Solution: Make sure SPIN is correctly installed, and that the directory containing the SPIN executable is part of your *path* environment variable. Perhaps you have renamed `spin` to `spin-linux` or `spin516`; maybe you do not have execute permission for SPIN, or perhaps you have another application called `spin` installed on your machine. Resolve these issues so that `spin` is all that needs to be typed to launch the SPIN tool.

6.1.12 C preprocessor, `cpp`, unavailable

Problem: TopSPIN uses the `cpp` program to expand `#define` and `#include` macros before processing a specification. If `cpp` is not available then TopSPIN can still be invoked, but cannot handle specifications which include these directives.

Example error message:

```
C preprocessor (cpp) not available - TopSPIN will not work correctly on files which use
#define or #include.
src.promela.lexer.LexerException: [1,1] Unknown token: #
    ... rest of Java stack trace
```

Solution: Install the `cpp` program on your system. Linux distributions provide `cpp` as standard; under Windows the `cpp` program is provided as part of Cygwin. Make sure `cpp` is in your path.

6.1.13 Typechecking error: problem with array index

Problem: By default, TopSPIN performs strict typechecking and only allows arrays to be accessed using `byte` or `pid` types.

Example error message:

```
Error at line 8: Type "int" cannot be used as an array index, it is not a subtype of "byte".
```

Solution: You can use the `-relaxedarrayindexing` option to make this error go away (see §3.2.3), or refactor your specification so that arrays are indeed only indexed by `byte` or `pid` expressions: this may make the associated state-vector smaller.

6.1.14 Typechecking error: problem with assignment to numeric variable

Problem: By default, TopSPIN performs strict typechecking and only allows a variable to be assigned a value of a "smaller" type, *e.g.* assignment of `byte` to `short` is OK, but not the other way round. Sometimes this is too restrictive.

Example error message:

```
Error at line 8: invalid assignment -- Type "short" occurs in a context where it is required to be a subtype o
```

Solution: You can use the `-relaxedassignment` option to make this error go away (see §3.2.4), or refactor your specification so that assignments to numeric variables are always from expressions with a smaller type: this may make the associated state-vector more compact.

6.1.15 Error during verification: "bad proctype"

Problem: Verification quits with a "bad proctype" error message.

Example error message:

```
pan: bad proctype - collapse (at depth 1023)
```

Solution: This error can occur if your specification involves proctypes which terminate. See §4.2 for details of this, and for a technique to prevent SPIN processes from terminating by providing an inescapable final state. This error message will only be generated if `COLLAPSE` compression is used. Nevertheless, for reasons explained in §4.2 it is best to apply TopSPIN to systems of non-terminating processes.

6.2 Reporting bugs in TopSPIN

If you have a problem using TopSPIN, first check to see if your problem is covered by one of the *common problems* in §6.1. If this is not the case, then please take the time to report your bug to the TopSPIN developers, so that it can be immediately documented and ultimately fixed, for the benefit of you and other TopSPIN users.

Bug reports should be submitted using the contact details in §6.4. Please provide the following when reporting a bug:

- A description of the problem
- A test-case (example Promela specification and `config.txt` file) which exposes the problem
- The TopSPIN version you are using, the approximate date on which you downloaded TopSPIN
- Details as to whether you are using a pre-compiled version of TopSPIN, or a version you have compiled from source
- Any ideas you have as to what may have gone wrong!

Please note that bugs do not have to relate to the correctness of TopSPIN – a valuable bug report could be in response to a misleading error message generated by TopSPIN, where there *is* something wrong with the input specification, but the problem is not what the error message indicates. Please feel free to also submit suggestions for features to be added to TopSPIN.

6.3 Reporting bugs in this manual

Please also get in touch (§6.4) if you find typos, inconsistencies or ambiguities in this manual – this kind of feedback is extremely valuable.

6.4 Getting in touch

All correspondence related to TopSPIN should be by email, to Alastair Donaldson: `ally@codeplay.com`.

Acknowledgements

- TopSPIN was originally developed by Alastair Donaldson at the University of Glasgow, between 2003 and 2006.
- TopSPIN uses symmetry reduction theory developed by Alastair Donaldson and Alice Miller. This theory is in turn based on a decade of research by the model-checking community.
- Thanks to the following people for reporting bugs in TopSPIN, and feedback on the instructions in this manual: Silver Juurik, Jan Rakow, Shamim Rippon, Iryna Shkvarko.
- Thanks to Gerard Holzmann, designer of SPIN, for advice on integrating symmetry reduction with other SPIN state-space reduction techniques.
- The approach taken by the TopSPIN implementation was influenced greatly by the SymmSpin tool, developed by Dragan Bosnacki, Denis Dams and Lezek Holenderski.
- The manual layout was inspired by the GAP reference manual.
- Thanks to the developers of GAP, *saucy* and SableCC: the development of TopSPIN would have taken a lot longer without these excellent tools.