HSF-SPIN User Manual

Institut für Informatik
Albert-Ludwigs-Universität
Georges-Köhler-Allee
D-79110 Freiburg
eMail: {edelkamp,lafuente}@informatik.uni-freiburg.de

March 22, 2006

Contents

1 Introduction

HSF-SPIN is a Promela model checker based on heuristic search strategies. It makes use of heuristic estimates and algorithms in order to direct the search into a specific error situation. As consequence it is able to find shorter trails than blind depth-first search. Moreover, directed search can sometimes find errors in larger state spaces. HSF-SPIN can be used for correctness verification, too, but this is not recommended, since other model checkers, like Spin, are more suitable.

HSF-SPIN was born from the fusion of the model checker Spin[1] and the heuristic search framework HSF[2]. It is basically an extension of HSF for searching state spaces generated by Promela models.

Like in Spin, two steps must be performed for running the verification. The first step is to generate the source code of the verifier for a given Promela specification. In the second step, the source is compiled and linked for constructing the verifier. The verifier is used for checking errors in the model. Among other parameters the user can specify the error type, the search algorithm, and the heuristic estimate as command line options. It is also possible to perform interactive or guided simulations similar to Spin. When the verification is done, statistic results are displayed and a solution trail in Spin's format is generated.

HSF-SPIN is based on Spin and its specification language Promela. Unfortunately, HSF-SPIN is not 100% Promela compatible. Promela specifications with dynamic or non deterministic process creation are not accepted in HSF-SPIN. This is due to the fact that such models rarely appear in practice. HSF-SPIN can check all the properties of Spin except from non-progress cycles. HSF-SPIN supports sequential bitstate hashing, but not partial order reduction. We are planning to study how to combine partial order reduction with heuristic search.

This document is intended to be a user guide of HSF-SPIN, containing the information needed to install and run HSF-SPIN. The reader is supposed to have some knowledge on protocol validation [**?**] and a user-level knowledge of Spin. The theoretical background of HSF-SPIN is described in [**?**] and [**?**]. Section **??** describes the installation process, Section **??** documents the compile- and runtime options of the tool and presents a first example, Section **??** deals with HSF-SPIN's output format, Section **??** give some guidelines for practical use of HSF-SPIN and Section **??** present a set of examples on using HSF-SPIN.

2 Installation

The first step of the installation process is to download a packed file from HSF-SPIN's home page[3] and unpack it. Supposing that the file is `HsfSpin1.0.tgz`, unpacking is done by typing:

```
user@host~ > tar -xzf HsfSpin1.0.tgz
```

[1] `http://netlib.bell-labs.com/netlib/spin/whatispin.html`
[2] `http://www.informatik.uni-freiburg.de/~edelkamp/Hsf/index.html`
[3] `http://www.informatik.uni-freiburg.de/~lafuente/hsf-spin`

HSF-SPIN is unpacked in a directory called HsfSpin. The next step is to compile the verifier generator, as when installing Spin. It is located in directory `hsf-spin`. The `makefile` must be edited. The file `README` in the directory provides more details.

```
user@host:~/HsfSpin > cd hsf-spin
user@host:~/HsfSpin/hsf-spin > emacs makefile
user@host:~/HsfSpin/hsf-spin > make
```

The executable `spin` is generated. It is very useful to have this command accessible from the *path* variable, and also to have a link to it with name `hsf-spin`. In the rest of the manual we will suppose this.

The next step is to edit the *makefile* in the directory `HsfSpin`. It is used for compiling the verifier. At the moment, it is only necessary to set the variable `TOPDIR` to value of the home directory.

```
user@host:~/HsfSpin/hsf-spin > cd ..
user@host:~/HsfSpin/hsf-spin > emacs makefile
```

3 Running HSF-SPIN

Verifying a model with HSF-SPIN is done in three steps. The first step is analogous to Spin, i.e., generating the sources of the verifier for a given Promela model. This is done with the compiled program `/HsfSpin/hsf-spin/Src3.4/spin`, which is assumed to be linked with `hsf-spin`. The second step is to compile the verifier, and the last step is to perform the verification. In the next subsection we present a first example. The following subsections deal with compile- and run-time options of HSF-SPIN.

3.1 A first example

The HSF-SPIN distribution includes a set of test models in the directory `models`. See the README file in this directory for more information about the models. File `deadlock.philosophers.prm` implements a Promela model of a deadlock solution to Dijkstra's dining philosophers problem. Let's start by generating the verifier for an 8-philosophers configuration of the model:

```
user@host:~/HsfSpin > hsf-spin -a -DN=8 models/deadlock.philosophers.prm
user@host:~/HsfSpin > make check
```

The executable `check` is a verifier of the model, similar to Spin's executable file `pan`. Deadlocks are checked by running the verifier with argument `-Ed`:

```
user@host:~/HsfSpin > check -Ed
HSF-SPIN 1.0
A Simple Promela Verifier based on Heuristic Search Strategies.
This tool is based on Spin 3.4.5  (by G.J. Holzmann) and
                  on HsfLight 2.0 (by S. Edelkamp)
```

```
Verifying models/deadlock.philosophers.prm...
Checking for deadlocks with Depth-First Search...
        invalid endstate (at depth 1362)
Printing Statistics...
        State-vector 120 bytes, depth reached 1362, errors: 1
             1341 states, stored
              431 states, matched
             1772 transitions (transitions performed)
               25 atomic steps
             1341 states, expanded
        Range of heuristic was: [0..0]
Writing Trail
Wrote models/deadlock.philosophers.prm.trail
        Length of trail is 1362
```

The verifier runs depth-first search, since it is the default search algorithm. It finds a deadlock at depth 1,362. Following such a long trail is tedious. Let's try to find a deadlock at a smaller depth with the A* algorithm (`-AA`) and a simple heuristic estimate for deadlock detection (`-Ha`):

```
user@host:~/HsfSpin > check -Ed -AA -Ha
HSF-SPIN 1.0
A Simple Promela Verifier based on Heuristic Search Strategies.
This tool is based on Spin 3.4.5  (by G.J. Holzmann) and
                  on HsfLight 2.0 (by S. Edelkamp)
Verifying models/deadlock.philosophers.prm...
Checking for deadlocks with A*...
        invalid endstate (at depth 34)
Printing Statistics...
        State-vector 120 bytes, depth reached 34, errors: 1
               67 states, stored
                6 states, matched
               73 transitions (transitions performed)
               25 atomic steps
               17 states, expanded
        Range of heuristic was: [0..8]
Writing Trail
Wrote models/deadlock.philosophers.prm.trail
        Length of trail is 34
```

A deadlock state has been now found at depth 34, expanding and storing less states, and performing less transitions. In the following we present a brief description of HSF-SPIN's compile- and run-time options. More examples are presented in Section **??**

3.2 Compile-Time Options

The verifier generator of HSF-SPIN generates a modified version of original Spin code.

The HSF-SPIN verifier accepts only a reduced subset of Spin's compile-time options, for example `-DVECTORSZ` and `-DGCC`. The only specific compile-time option is `-DDEBUG`, to report debug information when running.

3.3 Run-Time Options

Executing the HSF-SPIN verifier without arguments prints out all available run-time options:

```
user@host:~/HsfSpin > ./check
HSF-SPIN 1.0
A Simple Promela Verifier based on Heuristic Search Strategies.
This tool is based on Spin 3.4.5  (by G.J. Holzmann) and
                  on HsfLight 2.0 (by S. Edelkamp)
Parameters:
        -Ax: x=Search Algorithm
        -Ex: x=Error to be checked
        -Hx: x=Heuristic Function
        -Wx: x=Weighting for h in A*
        -wx: x=Weighting for g in A*
        -sx: 2^x=Size of Hash table (number of entries)
        -cx: 2^x=Size of Cache for IDA*
        -bx: 2^x=Size of BitState Hash table
        -Dx: x=Danger (for deadlock detection)
        -Sx: if x the program stops after search completed
        -rx: if x unreached states are printed
        -Px: if x solution is printed
        -dx: x=Depth bound
        -mx: x=Maximum value of heuristic estimate.
        -Rx: x=Number of traces (for Supertrace)
        -Tx: if x a trail file is created
```

Each argument of HSF-SPIN has the form $-Xy$, where X is the option to be set and y is the value for the option. For example, argument `-Ad` sets the option *search algorithm* to the value *depth-first search*. By giving an option no value, the list of available values for that option is printed. For example, executing `check -A` prints all available search algorithms:

```
user@hos:t~/HsfSpin/manual > ../check -A
HSF-SPIN 1.0
A Simple Promela Verifier based on Heuristic Search Strategies.
This tool is based on Spin 3.4.5  (by G.J. Holzmann) and
                  on HsfLight 2.0 (by S. Edelkamp)
Available Algorithms are:
        d: Depth-First Search (Hill Climbing if heuristic defined)
        A: AStar (Bread-First if no heuristic or wh=0. BestFirst if wg=0)
        i: IDAStar
        b: BitState IDAStar
        n: Nested depth-first search
```

```
        N: Improved Nested depth-first search
Parameters:
        -Ax: x=Search Algorithm
        -Ex: x=Error to be checked
        -Hx: x=Heuristic Function
        -Wx: x=Weighting for h in A*
        -wx: x=Weighting for g in A*
        -sx: x=Size of Hash table (number of entries)
        -cx: x=Size of Cache for IDA*
        -bx: x=Size of BitState Hash table
        -Dx: x=Danger (for deadlock detection)
        -Sx: if x the program stops after search completed
        -Px: if x solution is printed
        -dx: x=Depth bound
        -mx: x=Maximum value of heuristic estimate.
        -Rx: x=Number of traces (for Supertrace)
        -Tx: if x a trail file is created
```

The most important run-time options of HSF-SPIN are as follows.

Error to be checked Set with `-E`x, where x can take following values:

- `n`: The search has no goal; the verifier performs full state space exploration until the search is interrupted.
- `d`: The verifier searches for deadlocks. Deadlocks in Promela models are defined as non-valid endstates, i.e. endstates in which at least one of the processes of the model is in a state not marked as a valid endstate.
- `a`: The verifier searches for violation of assertions.
- `s`: The verifier searches for safety properties like deadlocks or assertion violations.
- `v`: The verifier searches for valid endstates.
- `l`: The verifier checks the violation of LTL properties. This includes bad sequences (claim violations) and bad cycles (accepting cycles).

Search Algorithm Set with `-Ax`, where x can take following values:

- `d`: Depth-first search. If the estimate is not the blind heuristic, successors are explored according to the heuristic value. In this case depth-first search performs Hill Climbing.
- `A`: A* . If the heuristic function is blind or the weight for the estimate is 0, then A* is in fact a breadth-first search. If the weight of the g value is zero, A* collapses to best-first search.
- `i`: Iterative deepening A* (IDA*).

- `b`: IDAStar with bitstate hashing. This algorithm is presented in [**?**].
- `n`: Classical nested depth-first search for detection of accepting cycles.
- `N`: Improved version of the nested depth-first search algorithm that exploits the structure of the never claim in order to close accepting cycles faster and to perform less transitions. The algorithm is presented in [**?**].

Heuristic Estimate Set with `-Hx`, where x can take following values:

- `t`: Guided search. A trail file will be followed similar to Spin's guided simulation.
- `i`: Interactive search. At each state expansion the user is prompted for the successor to explore similar to Spin's interactive simulation.
- `b`: Blind heuristic. It returns always 0. Used typically for performing a breadth-first search or a depth-first search.
- `e`: Number of enabled transitions (only for deadlocks).
- `a`: Number of non-blocked processes (only for deadlocks). In [**?**] this heuristic is called H_a.
- `f`: Formula based heuristic. Called H_f in [**?**]. (Only for safety errors).
- `c`: Distance to endstate in claim.
- `h`: Hamming distance with respect to a given goal state. The goal state is obtained from an error trail file.
- `s`: H_f with f being a formula that states that each process must be exactly in those local states that are defined by the error state.

User defined dangers for deadlock detection The heuristic estimate H_f for deadlock detection [**?**] is based on an inferred deadlock function f. With `-Dx` user aid is indicated to the verifier. Additionally, user aid can include the labels in the model. Labels with prefix `danger` help the inferring process to identify those states in which each process can block.

Weighting in A* A* expands states according with the f value of the state. The f value of a state is computed as the weighted sum of the distance g from the root and the estimate distance h to the goal state. In HSF-SPIN `-w`x sets the weight of g, while `-W`x sets the weight of h.

Depth bound With `-d`x the maximal reacheable depth is set to x

Maximum Heuristic Value The maximum value of the heuristic estimate can be set with `-m`x.

Number of Supertraces Usually, in sequential bitstate hashing with IDA*, several *runs* of the supertrace algorithm are performed. The number of *runs* is set in HSF-SPIN with `-R`x.

Size of Hash Table The number of entries (as power of 2) of the hash table used for allocating the visited states in various search algorithms can be set with `-s`x.

Size of Cache for IDA* IDA* works with a cache, which size (as power of 2) can be set with `-c`x.

Size of Bitstate Hash Table The size (as power of 2) of the bitstate hash table is set with `-b`x

Writing the Error Trail By default an error trail file is written if an error is found. You can avoid this by setting `T0`.

4 HSF-SPIN's Output Format

The HSF-SPIN verifier provides statistic results, and report solution trails if it finds an error.

Statistics results produced by HSF-SPIN are a little different from those of Spin. While in Spin the most important statistic data are the number of stored states (space consumption) and the number of transitions performed (time consumption), in HSF-SPIN the number of expanded nodes is also important. Thus at the end of the verification the HSF-SPIN verifier outputs these three statistical parameters, as well as the depth reached, the size of the state vector and the range of the heuristic estimate. The verifier also outputs these parameters, when the user interrupts the search, and every 100,000 memorized states (as a snapshot of the search). When using IDA*, the number of generated, stored and expanded states are displayed for each threshold. At the end the total number of transitions and expanded states are displayed, but the displayed number of stored states corresponds only to the last threshold.

HSF-SPIN provides error trails in Spin's format. This allows to post-process the trail with Spin's graphical front-end XSpin. The verifier can also output the trail in an own textual format if it is executed with `-P1`.

4.1 Improved NDFS in Spin

The HSF-SPIN distribution includes an experimental extension of Spin that supports the improved nested depth-first search algorithm. To install it correctly, the sources in `indfs-spin` must be compiled exactly as in Spin. Only if the pan sources are compile with `-DINDFS`, the improved algorithm is used instead of the original one. It requires one of the files contained in the `include` directory, so we recommend to use the makefile provided by HSF-SPIN.

4.2 Analyzing Strongly Connected Components

The HSF-SPIN distribution includes also a program that classifies the strongly connected component of the state transition graph of the never claim and the processes of the model. Once the pan files are generated with `hsf-spin`, `make scc` constructs this program.

5 Practical use of HSF-SPIN

HSF-SPIN is an experimental tool developed for research purposes. Nevertheless, we believe that our tool can be used in practice for finding shorter error trails than Spin, and also for finding errors where Spin fails. Since HSF-SPIN offers a wide variety of search strategies, it is not always clear how to use it. In this section we provide a guideline for using it.

If Spin is able to find a safety error, but the error trail is too long to comprehend, HSF-SPIN can be used for finding a better trail. This is done by using one of the heuristics for known errors, the hamming distance for example, and A* or one of its derivates. HSF-SPIN will try to reproduce the error trail and to use it for finding a shorter trail to error state. Section **??** show an example for this.

If Spin is not able to find a safety error, then HSF-SPIN can be used for finding it. The error to be checked and the search strategy to be used must be specified. The selection of the search algorithm is a tradeoff between space requirements and solution quality. The depth-first search algorithm of HSF-SPIN should be avoided, since it is notably less inefficient than that of Spin. Weighting A* is a tradeoff between space requirements and solution quality. In general best-first requires less memory and time but offers larger trails, while breadth first offers optimal trails, but its memory consumption is maximal. IDA* and its bitstate derivate are time consuming and should be used in case of exhausted space resources only.

Checking liveness properties such as accepting cycles, should be done with the extension of Spin that includes the improved nested depth-first search algorithm. Section **??** show some example for this.

6 Examples

In this section we present some examples on using HSF-SPIN.

Getting a shorter trail Let's suppose we are trying to find a deadlock in the philosophers model using Spin:

```
user@host: ~/HsfSpin > spin -a -DN=8 models/deadlock.philosophers.prm
user@host: ~/HsfSpin > make pan
user@host: ~/HsfSpin > pan
pan: invalid endstate (at depth 1362)
```

```
pan: wrote models/deadlock.philosophers.prm.trail
(Spin Version 3.4.5 -- 8 March 2000)
Warning: Search not completed

Full statespace search for:
        never-claim             - (none specified)
        assertion violations    +
        acceptance   cycles     - (not selected)
        invalid endstates       +

State-vector 120 byte, depth reached 1362, errors: 1
    1341 states, stored
     431 states, matched
    1772 transitions (= stored+matched)
      25 atomic steps
hash conflicts: 0 (resolved)
(max size 2^18 states)

1.868   memory usage (Mbyte)
```

Spin finds a deadlock state at depth 1,362. Suppose now that we want to use HSF-SPIN to find the deadlock state at a smaller depth. HSF-SPIN offers the possibility of reproducing an error state and using it as information for guiding the search. For example, we can perform a search for the deadlock state with A* as the search algorithm and the hamming distance as the heuristic estimate:

```
user@host: ~/HsfSpin > hsf-spin -a -DN=8 models/deadlock.philosophers.prm
user@host: ~/HsfSpin > make check
user@host: ~/HsfSpin > check -Ed -AA -Hh
HSF-SPIN 1.0
A Simple Promela Verifier based on Heuristic Search Strategies.
This tool is based on Spin 3.4.5  (by G.J. Holzmann) and
                  on HsfLight 2.0 (by S. Edelkamp)
Verifying models/deadlock.philosophers.prm...
Opening trail file models/deadlock.philosophers.prm.trail...
Trying to reproduce trail...
        invalid endstate (at depth 1362)
Trail reproduced.
Checking for deadlocks with A*...
        invalid endstate (at depth 34)
Printing Statistics...
        State-vector 120 bytes, depth reached 34, errors: 1
               45 states, stored
                0 states, matched
               45 transitions (transitions performed)
               25 atomic steps
               10 states, expanded
        Range of heuristic was: [0..192]
Writing Trail
Wrote models/deadlock.philosophers.prm.trail
```

```
        Length of trail is 34
```

As we can see, HSF-SPIN finds in fact a shorter error trail.

Improved Nested Depth-First Search HSF-SPIN includes an improved version of the nested depth-first search algorithm for checking acceptance cycles. This algorithm has been implemented in both HSF-SPIN and SPIN. We will use the model `ltl.elevator.prm` as test case. This model does not satisfy a certain response property. Let's begin trying to check the property with the original nested depth-first search algorithm:

```
user@host:~/HsfSpin > hsf-spin -a -DN=3 models/ltl.elevator.prm
spin: unexpected: seqno=19 ntyp=46 (18 ntyp=275)
spin: warning: bit-array level_requested[2] mapped to byte-array
spin: warning: bit-array level_requested[2] mapped to byte-array
user@host:~/HsfSpin > make check
user@host:~/HsfSpin > check -El -An
HSF-SPIN 1.0
A Simple Promela Verifier based on Heuristic Search Strategies.
This tool is based on Spin 3.4.5  (by G.J. Holzmann) and
                  on HsfLight 2.0 (by S. Edelkamp)
Verifying models/ltl.elevator.prm...
Checking for violation of an LTL formula with Nested Depth First Search...
        acceptance cycle in dfs2 (at depth 255, length 101)
Printing Statistics...
        State-vector 72 bytes, depth reached 355, errors: 1
              224 states, stored
               44 states, matched
              348 transitions (transitions performed)
                9 atomic steps
              275 states, expanded
        Range of heuristic was: [0..0]
Writing Trail
Wrote models/ltl.elevator.prm.trail
        Length of trail is 356
```

Now let's see how to used the improved algorithm. HSF-SPIN's verifier can be used by executing `check -El -AN`:

```
user@host:~/HsfSpin > check -El -AN
HSF-SPIN 1.0
A Simple Promela Verifier based on Heuristic Search Strategies.
This tool is based on Spin 3.4.5  (by G.J. Holzmann) and
                  on HsfLight 2.0 (by S. Edelkamp)
Verifying models/ltl.elevator.prm...
Checking for violation of an LTL formula with Improved Nested Depth First Search...
        acceptance cycle in dfs1 (at depth 241, length 102)
Printing Statistics...
        State-vector 72 bytes, depth reached 341, errors: 1
```

```
             217 states, stored
              38 states, matched
             255 transitions (transitions performed)
               9 atomic steps
             217 states, expanded
        Range of heuristic was: [0..0]
Writing Trail
Wrote models/ltl.elevator.prm.trail
        Length of trail is 343
```

Now we show how to do the same, but with the Spin extension that supports the improved nested depth-first search:

```
user@host~/HsfSpin > indfs-spin/Src3.4/spin -a -DN=3 models/ltl.elevator.prm
spin: unexpected: seqno=19 ntyp=46 (18 ntyp=275)
spin: warning: bit-array level_requested[3] mapped to byte-array
user@host~/HsfSpin > make ipan
user@host~/HsfSpin > ./ipan
warning: for p.o. reduction to be valid the never claim must be stutter-closed
(never claims generated from LTL formulae are stutter-closed)
pan: acceptance cycle in dfs1 (at depth 241)
pan: wrote models/ltl.elevator.prm.trail
(Spin Version 3.4.5 -- 8 March 2000)
Warning: Search not completed
        + Partial Order Reduction

Full statespace search for:
        never-claim             +
        assertion violations    + (if within scope of claim)
        acceptance   cycles     + (fairness disabled)
        invalid endstates       - (disabled by never-claim)

State-vector 72 byte, depth reached 342, errors: 1
     214 states, stored
      31 states, matched
     245 transitions (= stored+matched)
       9 atomic steps
hash conflicts: 0 (resolved)
(max size 2^18 states)

1.868   memory usage (Mbyte)
```

7 Contact

HSF-SPIN is being developed by Stefan Edelkamp and Alberto Lluch Lafuente at the Institute of Computer Science, Albert-Ludwig-Universität Freiburg, Germany. We would be grateful to receive any kind of feedback. Emails can be send to Alberto Lluch Lafuente (lafuente@informatik.uni-freiburg.de). Further

information on HSF-SPIN can be found at HSF-SPIN's home page[4]. Information on research using HSF-SPIN can be found at our site on directed model checking[5].

References

[1] S. Edelkamp, A. L. Lafuente, and S. Leue. Protocol verification with heuristic search. In *AAAI Symposium on Model-based Validation of Intelligence*, 2001.

[2] A. L. L. Stefan Edelkamp and S. Leue. Directed explicit model checking with hsf-spin. In *8th International SPIN Workshop on Model Checking Software*, 2001.

[4] `http://www.informatik.uni-freiburg.de/~lafuente/hsfspin/index.html`
[5] `http://www.informatik.uni-freiburg.de/~leue/dirmodcheck.html`