NAME

smv – symbolic model verifier

SYNOPSIS

smv [*options*] [*input-file*]

DESCRIPTION

smv is a program that uses a symbolic model checking algorithm to evaluate formulas of **CTL** (**C**omputational **T**ree **L**ogic – a branching time temporal logic) with respect to a finite state model. The model and the specifications are described in *input-file* (default is the standard input). The language for describing the model is a simple parallel assignment. For complete definition of **CTL** and the model description language, please refer to the document "The SMV system."

OPTIONS

–AG Verify **Universal CTL** formulas only.

–c *cache-size*

Set the size of the cache for BDD operations. It should be a prime or nearly prime number. Default is 16381. There is a tradeoff here between performance and memory. Up to a point, a larger cache can speed up operations by orders of magnitude. Each cache entry uses 16 bytes, so a quarter million entries use about four megabytes, which is reasonable if you have about 12 megabytes of real memory available. Virtual memory is of practically no use.

Some suggested values for the -c parameter:

16381, 65521, 262063, 522713, 1046429

–cp *part-limit*

Perform conjunctional partitioning of the transition relation. The transition relation is not evaluated as a whole, instead the various **next(***variable***)** assignments are collected into partitions. When the size of a partition exceeds *part-limit*, the remaining assignments are collected into a new partition. When a forward (or backward) traversal of the transition relation is needed, each partition is used in turn. After the use of each partition, early quantification is done on unnecessary variables in order to reduce the size of the intermediate BDD [This option currently makes **smv** run slower or use more memory].

–f Search the reachable state space of the model before evaluating the **CTL** operators. This can result in improved performance for models with sparse state spaces.

–h *heuristic-factor*

The variable ordering is determined by a heuristic procedure which is based on the syntactic structure of the program, and a floating point *heuristic-factor* between 0.0 and 1.0 [This option is currently broken].

–i *input-order*

The variable ordering is read from file *input-order*. This overrides the **–h** option. This is most useful in combination with the **–o** option: The variable ordering (with or without heuristic ordering) can be written to a file using the **–o** option, the file can be inspected and reordered by the user, then read back in using the **–i** option. See **VARIABLE ORDERING** below.

–inc Perform incremental evaluation of the transition relation. At each step in the forward search, the transition relation is being restricted to the reached state set. This can cut down on the size of the transition relation, although at the expense of some overhead to reevaluate at each step.

–int **smv** enters interactive mode after the processing of *input-file* is completed. See **INTERACTIVE MODE** below.

–k *key-table-size*

Set the size of the key table for BDD nodes. It should be a prime or nearly prime number, and should be at least 1/10 the number of BDD nodes in use at any given time. The default is

16381, which should be enough for most applications.

–m *mini-cache-size*
Sets the size of the portion of the cache for BDD operations which is used by the less expensive (non-iterative) BDD operations. It should be a prime or nearly prime number not larger than the *cache-size*. The default is 16381, same as the default *cache-size*.

–o *output-order*
The variable ordering is written to file *output-order*, after parsing, and optional application of the heuristic variable ordering procedure (**–h**). No evaluation occurs when the **–o** option is used.

–r The number of reachable states to be printed at termination. This can involve some extra work if the **–f** option is not used.

–v *verbose-level*
A large amount of gibberish printed on the standard error. Setting *verbose-level* to 1 should give you all the information you need. Using this option makes you feel better, since otherwise the program prints nothing until it finishes, and there is no evidence that it is doing anything at all. Setting the *verbose-level* higher than 2 has the same affect as 2.

VARIABLE ORDERING

smv uses Boolean Decision Diagrams (BDDs) to represent sets and relations in the **CTL** model checking algorithm. A BDD is a decision tree, in which variables always appear in the same order as the tree is traversed from root to leaf. The efficiency of BDDs is obtained by always combining isomorphic subtrees, and by eliminating redundant decision nodes in the tree. The degree storage efficiency obtained in this way is closely related to the variable ordering. The present version of the program has no built-in heuristics for selecting the variable ordering. Instead, the variables appear in the BDDs in the same order in which they are declared in the program. This means that variables declared in the same module are grouped together, which is generally a good practice, but this alone is not generally sufficient to obtain good performance in the evaluation. Usually, the variable ordering must be adjusted by hand, in an ad hoc way. A good heuristic is to arrange the ordering so that variables which often appear close to each other in formulas are close together in the order of declaration, and global variables should appear first in the program. The number of BDD nodes currently in use is printed on standard error each time the program performs garbage collection, if verbose-level is greater than zero. Also, as each evaluation is made, the number of BDD nodes representing the result is printed. An important number to look at is the number of BDD nodes representing the transition relation. It is very important to minimize this number. Iterations are used to solved the fixed point equations which characterize the **CTL** operators, and also to search for counterexamples. With each iteration, the number of BDD nodes used to represent the result is printed, as well as the number of corresponding states. Some of the *options* can improve performance. Experiment with them if the run time starts getting out of hand.

INTERACTIVE MODE

When the **–int** option is used, **smv** goes into interactive mode after the specifications in *input-file* has been checked. In this mode, the model described in *input-file* is used as a basis for interactive debugging and modifications. Moreover, specific states of the model can be reached using any trace created by **smv** in either interactive or non-interactive mode. A trace is a sequence of states corresponding to a possible execution of the model. Each trace produced by **smv** has a number, and the states are numbered within the trace. Trace number *n* has states numbered *n.1, n.2, n.3,* . . . If additional traces are needed, say from state *n.i*, these traces are numbered *n.i.1, n.i.2, n.i.3,* . . . Within these traces, the states are numbered *n.i.m.1, n.i.m.2, n.i.m.3,* . . . In the interactive mode **smv** associates a *current state* with one of the states of the model. Most of the commands operate on the *current state*. The *current trace* is the trace the *current state* belongs to.

Interactive Commands

The following commands are recognized in interactive mode:

EVAL *expression* **;**
expression is evaluated in the *current state*. *expression* may be a **CTL** formula, and therefore, can produce a trace, from *current state*, to be used by later commands.

FAIR *expression* **;**
Add a new fairness constraint to the existing list of fairness constraints (See "The SMV System").

GOTO *state* **;**
Make *state* the *current state*.

INIT *expression* **;**
Add a constraint on the initial states. *expression* should hold for all initial states. This command is equivalent to the **INIT** declaration in *input-file* (See "The SMV System").

LET *variable* **:=** *expression* **;**
Assign the value of *expression*, as evaluated in the *current state*, to *variable*. This command changes the *current state*. The value of all other variables in the new *current state* remains the same as it was in the old *current state*.

RESET ;
Discard all additions made to the model in interactive mode. This command cancels the effect of all **FAIR**, **INIT**, and **TRANS** commands issued in interactive mode.

SPEC *expression* **;**
The specification *expression* is evaluated in all of the initial states. This command is equivalent to the **SPEC** declaration in the *input-file*.

STEP ; Move to the next state in the current trace.

TRANS *expression* **;**
Add *expression* to the constraints on the transition relation. This command is equivalent to the **TRANS** declaration in the *input-file* (See "The SMV System").

SEE ALSO

The SMV system,
Symbolic Model Checking – an approach by K. McMillan, CMU-CS-92-131

BUGS

Arguments of the wrong type specified for certain *options* and *commands* may produce cryptic (and fatal) error messages.

AUTHOR

Kenneth L. McMillan, Carnegie Mellon University.
mcmillan@cs.cmu.edu