NAME

verify -- model-check a system of communicating machines

SYNOPSIS

verify [*option* ...] [*scm-file*]

DESCRIPTION

The **verify** tool reads the model given in the specified *scm-file* and model-checks it. If *scm-file* is omitted, the model is read from the standard input.

The model contains the description of a system of communicating machines along with, optionally, the specification of a set of bad configurations. See scm(5) for a description of the scm input language.

Upon succesful termination, **verify** either reports that the model is safe (if no bad configuration is reachable), or provides a counter-example trace (leading to some bad configuration).

Note that, depending on the input model and on the options that are used, **verify** may abort, or compute indefinitely. This problem cannot be avoided due to the undecidability of the model-checking problem for systems of communicating machines.

OPTIONS

Main Options

-mc-engine *engine*

Specifies the main algorithm used for model-checking. The following *engines* are available:

absint Static analysis based on abstract interpretation.

armc Abstract regular model-checking.

cegar Counter-example guided abstraction refinement. This algorithm maintains an abstract graph whose nodes form a partition of the model's configuration space. Refinement consists in splitting abstract nodes visited by a spurious counter-example, according to a generated trace invariant. This is the default.

lart Lazy abstraction refinement tree. This algorithm builds an abstract reachability tree, and maintains node covering information to truncate the tree. Branches leading to bad configurations are refined by strengthening nodes according to a generated trace invariant.

-tc-engine *engine*

Specifies the trace-checking algorithm used by some model-checkers. Here, trace-checking consists in analyzing a given abstract counter-example to determine whether it is feasible or

spurious, and, in the latter case, provide a *trace invariant* ‘‘explaining’’ spuriousness. The following *engines* are available:

apinv-bwd Perform an adaptive extrapolated symbolic *pre* computation along the abstract counter-example, starting from the failure abstract node. This is the default.

apinv-fwd Reversed version of **apinv-bwd** (uses *post*).

upinv-fwd Perform a uniform extrapolated symbolic *post* computation along the abstract counter-example.

upinv-bwd Reversed version of **upinv-fwd** (uses *pre*).
This option only impacts the **cegar** model-checking engine.

-k-min *start*

The extrapolation operator is parameterized by a non-negative integer *k*. This option specifies the starting value of *k*. The model/trace-checking algorithms first try with *k* = *start* (which produces the coarsest over-approximation) and then iteratively increase *k* until the analysis is precise enough (or *k* becomes greater than the bound specified by **-k-max**). The *start* must be a nonnegative integer. The default is 0.

As an exception (a hack), it is also possible to specify a *start* of -1 (see option **-extrapolation** below).

-k-max *stop*

Specifies the largest value that model/trace-checking algorithms may use for the extrapolation parameter *k*. The *stop* must be a nonnegative integer. The default is 2.

-tc-discard-states

Removes abstract states from the trace before passing it to the trace-checking engine. With this option, abstract counter-examples are more likely to be feasible. But, for spurious counter-examples, trace invariant generation may require more precision (i.e., a higher extrapolation parameter).

-tc-validate

Validates each result provided by the trace-checking engine, and print an error message when validation fails.

-limit *limit*

Limit the number of model-checker iterations to *limit*. Note that iterations are not comparable between model-checking engines. The *limit* must be a nonnegative integer. OCaml’s max_int is used by default.

-no-validation

The model-checking *engine* returns either a safe inductive invariant, or a counter-example trace. By default, this result is validated by a simple, independent algorithm. This option disables this validation.

-statistics

Upon succesful termination, **verify** displays detailed statistics about execution time and memory consumption.

Scm Wrapper Options

-scm-rel *method*

The symbolic one-step binary reachability relation of the model is not (directly) provided by the *Scm* library. This option specifies the method used to compute this binary relation. The following *methods* are supported:

post Use the symbolic *post* operator of the *Scm* library.

pre Use the symbolic *pre* operator of the *Scm* library.

mixed Use *post* for send actions and *pre* for receive actions. This is the default.

-extrapolation *operator*

Specifies the extrapolation operator used for over-approximation of queue decision diagrams. The over-approximations provided by the *Scm* library are based on state equivalences. The following extrapolation *operators* are available:

bisim-fwd Forward *k*-depth bisimulation equivalence. This is the default.

bisim-bwd Backward *k*-depth bisimulation equivalence.

bisim-both Conjunction of the two previous equivalences.

lang-fwd Forward *k*-depth language equivalence.

lang-bwd Backward *k*-depth language equivalence.

lang-both Conjunction of the two previous equivalences.

identity Identity function (no over-approximation).

As an exception, the extrapolation for $k = -1$ is the same for all *operators*, and consists in merging all states of the queue decision diagram. In other words, this extrapolation is equal to the 0-depth bisimulation/language equivalence without state coloring.

-extrapol-coloring *coloring*

Specifies the state coloring applied by the extrapolation operator on queue decision diagrams. The state equivalence induced by this coloring defines the 0-depth equivalence (see **-extrapolation**). The following state *colorings* are available:

final Distinguish final states from the others (2 colors).

init Distinguish initial states from the others (2 colors).

both Conjunction of the two previous equivalences (4 colors).

none Merge all states (1 color).

auto Behave as **final** for forward extrapolations, as **init** for backward extrapolations, and as **both** for extrapolations that use both directions. This is the default.

-channel-policy *policy*

Channels in an *scm* model can be either perfect or lossy (see scm(5)). This option allows to override the reliability policy of the channels. The following reliability *policies* are supported:

all-lossy Make all channels lossy.

all-perfect Make all channels perfect.

normal Keep the policy specified in the *scm* input. This is the default.

Cegar Options

-graph-exploration *search*

Specifies the exploration algorithm used to find a counter-example in the abstract graph. The following graph *search* algorithms are supported:

fwd-dfs Forward depth-first search.

fwd-bfs Forward breadth-first search.

bwd-dfs Backward depth-first search.

bwd-bfs Backward breadth-first search.

mixed Use only breadth-first search and select the direction with the smallest set of initial nodes. This is the default.

-graph-refinement *method*

When an abstract node is split into *n* refined nodes, the abstract edges that enter or leave this node must be refined. This option specifies the method used to compute the refined edges. The following *methods* are supported:

rel Use the model's symbolic one-step binary reachability relation *rel*. This method requires, for each edge refinement, *n rel* computations.

post-pre Use the model's symbolic *post* and *pre* operators. This method requires, for each edge refinement, 1 *post*/*pre* computation and *n* emptiness tests. This is the default.

Lazy Abstraction Options

-tree-exploration *exploration*

Specifies the exploration strategy used to build the abstract reachability tree. The following *exploration* strategies are supported:

dfs Depth-first exploration.

bfs Breadth-first exploration. This is the default.

-no-tree-pruning

By default, nodes being refined with an empty region are removed from the tree. This option disables this pruning.

APInv Options

-safe-approximation *algorithm*

Adaptive trace invariant generation relies on a simplification procedure that, given a pair (*r1*, *r2*) of disjoint regions, returns an over-approximation of *r1* that is still disjoint from *r2*. This option selects the algorithm used for the computation of this safe over-approximation. The following *algorithms* are available:

split Given (*r1*, *r2*), return the extrapolation of *r1* for the smallest parameter *k* that leads to an empty intersection with *r2*. This is the default.

co-split Given (*r1*, *r2*), return the complement of the region obtained with **split** on (*r2*, *r1*).

Logging Options

-columns *width*

Use *width* columns for pretty-printing. In other words, set the value of the pretty-printer's right margin to *width* (in characters). The default is the value of the environment variable COLUMNS if set, or 78 otherwise.

-box-depth *depth*

Set the pretty-printer's box depth to *depth*. Boxes nested deeper than *depth* are printed as an ellipsis. OCaml's max_int is used by default.

SEE ALSO

control(1), scm(5)

AUTHORS

The **verify** tool is mainly written, maintained and tested by:

Alexander Heussner <*alexander.heussner@labri.fr*>
Gregoire Sutre <*gregoire.sutre@labri.fr*>

See the *AUTHORS* file in the source distribution for the full list of contributors.

Web site: *http://altarica.labri.fr/forge/projects/mcscm/wiki*

CAVEATS

Numerical values held by messages or local variables are currently ignored by **verify**.

- Conditions in transitions and bad configurations are discarded and assumed to be **true**.

- Assignments in transitions are ignored.

These rules guarantee that the analysis performed by **verify** is ‘‘safety-conservative’’.

BUGS

Please visit the following page for bug reports and feature requests:
http://altarica.labri.fr/forge/projects/mcscm/issues

ACKNOWLEDGEMENTS

The **verify** tool is programmed in *Objective Caml* and uses several 3rd-party libraries:

- *Camllib*, *Fixpoint*, *LatticeAutomata*, and *Scm*. By Bertrand Jeannet and Tristan Le Gall. Licensed under the LGPL. Web site: *http://gforge.inria.fr/projects/bjeannet/*.