NAME

mcscm - model-check a system of communicating machines

SYNOPSIS

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

DESCRIPTION

The **mcscm** 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 the *SCM LANGUAGE* section for more details.

Upon succesfull termination, **mcscm** 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, **mcscm** 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 partition abstraction refinement. This is the default.

-no-verification

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

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).

-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 LANGUAGE*). 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 vertices. This is the default.

-partition-refinement *method*

When the abstract graph contains a spurious counter-example, the underlying partition is refined in order to eliminate this counter-example. Refinement consists in splitting some abstract nodes visited by the spurious counter-example, and relies on a *trace invariant* that ‘‘explains’’ why the abstract counter-example is spurious. This option specifies the method used to compute trace invariants. The following *methods* 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*).

-extrapol-start-param *start*

The extrapolation operator is parameterized by a non-negative integer *k*. This option specifies the starting value of *k*. The trace invariant generation algorithms first try with *k* = *start* (which produces the coarsest over-approximation) and then iteratively increases *k* until

a precise enough refinement is obtained. The *start* must be a nonnegative integer. The default is 0.

As an exception, it is also possible to specify a *start* of -1. The extrapolation for k = -1 consists in merging all states of the queue decision diagram. In other words, this extrapolation is equal to the bisimulation-based extrapolation for k = 0 with **-extrapol-coloring none**.

-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.

-limit *limit*

Limit the number of Cegar loop iterations to *limit*. The *limit* must be a nonnegative integer. The default is OCaml's max_int.

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*).

Miscellaneous Options

-screen-width *width*

Use *width* columns for pretty-printing.

SCM LANGUAGE

The *scm* textual language was introduced in the Ph.D. thesis of Tristan Le Gall. An *scm* model contains the description of a system of communicating machines. This description is composed of two parts:

- A *header* that specifies the set of fifo channels and the message alphabet.

- A list of *automata*, each modeling a communicating machine.

For model-checking purposes, the format also permits the specification of a set of *bad configurations*.

Header

1. Start the description of a system of communicating machines and specify its name:

   scm *ident* **:**

2. Specify the number of channels:

   nb_channels = *integer* **;**

   The set of channels is {0, ..., *n*-1} where *n* is the number of channels. By default, all channels are perfect.

3. Tag some channels as lossy (optional):

   [**lossy :** *integer* [**,** *integer* ...]]

4. Declare the message alphabet as follows:

   parameters : [{**int** | **real**} *ident* [= *expr*] **;** ...]

   The alphabet is the same for all channels. Each message holds a typed numerical value.

Automata

1. Start the description of an automaton and specify its name:

   automaton *ident* **:**

2. Declare the automaton's local variables (optional):

   [{**int** | **real**} *ident* [= *expr*] **;** ...]

3. Specify the automaton's initial states:

   initial : *integer* [**,** *integer* ...]

4. Declare the automaton's states together with outgoing transitions:

state *integer* **:** [**to** *integer* **:** *command* **;** ...]

where *command* is of the following form:

when *cond* [**,** *integer* {**!** | **?**} *ident*] [**with** *ident* **=** *expr* [**,** *ident* **=** *expr* ...]]

Semantics

The operational semantics of a system of communicating machines is the usual one: the automata move asynchronously according to their local transitions, and they communicate exclusively through the channels. Communication actions are **!** (send) and **?** (receive). Channels are fifo, unbounded, and initially empty. Note that channels need not be point-to-point, they are shared by all automata.

Bad Configurations

The description of the system of communicating machines is, optionally, followed by the specification of a set of bad configurations:

bad_states : (automaton *ident* **:** *badlocal* ... [**with** *regexp*] **)** ...

where *badlocal* is of the following form:

in *integer* **:** *cond* ...

The local constraints in a *badlocal* specification are *disjuncted* together. The *badlocal* specifications in a **bad_states** declaration are *conjuncted* together.

The symbol **#** is used to separate channels in *regexp*. Each word matched by *regexp* must contain exactly *n*-1 occurrences of **#** where *n* is the number of channels.

AUTHORS

The **mcscm** 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 **mcscm**.

- 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 **mcscm** 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 **mcscm** 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/*.