NAME

scm -- scm language

DESCRIPTION

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 queue contents are represented by a regular expression *regexp* as defined below.

Regular Expressions

Queue contents for specifying bad states or displaying the current states of a system of communicating machines are represented by regular expressions of the following form:

regexp
::= _ | **#** | *a* for *a* in the message alphabet
| *regexp* **.** *regexp* | **(** *regexp* | *regexp* **)**
| *regexp* **^*** | *regexp* **^+**

The operators on regular expressions are interpreted as usually: **.** stands for concatenation, **|** for disjunction, **^*** is the reflexive-transitive closure, and **^+** the transitive closure. The empty word is denoted by _ .

The symbol **#** is interpreted as separator for channels in *regexp*. Each word matched by *regexp* must contain exactly *n*-1 occurrences of **#** where *n* is the number of channels.

EXAMPLE

The following example is the classical connection/disconnection protocol.

```
scm connection_disconnection :
nb_channels = 2 ;
parameters :
  real o ;
  real c ;
  real d ;
automaton sender :
  initial : 0
  state 0 :
    to 1 : when true , 0 ! o ;
  state 1 :
    to 0 : when true , 0 ! c ;
    to 0 : when true , 1 ? d ;
automaton receiver :
  initial : 0
  state 0 :
    to 1 : when true , 0 ? o ;
  state 1 :
    to 0 : when true , 0 ? c ;
    to 0 : when true , 1 ! d ;
bad_states:
  (automaton receiver: in 0 : true with c.(o|c)^*.#.d^*)
```

This *scm* model has two channels, with message alphabet {*o*, *c*, *d*}. There are two automata, a sender and a receiver. Their states and transitions should be self-explanatory. Bad configurations are those where (a) the receiver is in local state 0, and (b) channel contents are in the regular expression detailed below.

The regular expression *c* **.(** *o* **|** *c* **)^*. # .** *d* **^*** is interpreted as the set of contents of two channels:

- The first containing any word over the alphabet {*o*, *c*} that starts with a message *c*.

- The second containing an arbitrary sequence of *d* (which can be of zero length, i.e., empty).

SEE ALSO

control(1), verify(1)