Class SM

Known Subclasses:

Cascade, Constant, R, Feedback, FeedbackAdd, FeedbackSubtract, Gain, If, Switch, Parallel, PureFunction, Repeat, RepeatUntil, Select, Sequence, Until, Wire

---

Generic superclass representing state machines. Don't instantiate this: make a subclass with definitions for the following methods:

- `getNextValues: (state_t, inp_t) -> (state_t+1, output_t)` or `getNextState: (state_t, inpt_t) -> state_t+1`
- `startState: state` or `startState() -> state`

optional:

- `done: (state) -> boolean` (defaults to always false)
- `legalInputs: list(inp)`

See State Machines chapter in 6.01 Readings for detailed explanation.

Instance Methods

	getStartState(self) Handles the case that self.startState is a function.
	getNextValues(self, state, inp) Default version of this method.
	done(self, state) By default, machines don't terminate
	isDone(self) Should only be used by transduce.
	start(self, traceTasks=[], verbose=False, compact=True, printInput=True) Call before providing inp to a machine, or to reset it.
	step(self, inp) Execute one 'step' of the machine, by propagating `inp` through to get a result, then updating `self.state`.
	transduce(self, inps, verbose=False, traceTasks=[], compact=True, printInput=True, check=False) Start the machine fresh, and feed a sequence of values into the machine, collecting the sequence of outputs
	run(self, n=10, verbose=False, traceTasks=[], compact=True, printInput=True, check=False) For a machine that doesn't consume input (e.g., one made with `feedback`, for `n` steps or until it terminates.
	transduceF(self, inpFn, n=10, verbose=False, traceTasks=[], compact=True, printInput=True) Like `transduce`, but rather than getting inputs from a list of values, get them by calling a function with the input index as the argument.
	guaranteeName(self) Makes sure that this instance has a unique name that can be used for tracing.
	printDebugInfo(self, depth, state, nextState, inp, out, debugParams)

	Default method for printing out all of the debugging information for a primitive machine.
	`doTraceTasks(self, inp, state, out, debugParams)` Actually execute the trace tasks.
	`check(thesm, inps=None)` Run a rudimentary check on a state machine, using the list of inputs provided.

Class Variables

	`startState` = None By default, startState is none
	`legalInputs` = [] By default, the space of legal inputs is not defined.
	`name` = None Name used for tracing

Instance Variables

	`state` Instance variable set by start, and updated by step; should not be managed by user

Method Details

getStartState(self)

Handles the case that self.startState is a function. Necessary for stochastic state machines. Ignore otherwise.

getNextValues(self, state, inp)

Default version of this method. If a subclass only defines `getNextState`, then we assume that the output of the machine is the same as its next state.

isDone(self)

Should only be used by transduce. Don't call this.

start(self, traceTasks=[], verbose=False, compact=True, printInput=True)

Call before providing inp to a machine, or to reset it. Sets self.state and arranges things for tracing and debugging.

Parameters:

- **`traceTasks`** - list of trace tasks. See documentation for `doTraceTasks` for details
- **`verbose`** - If `True`, print a description of each step of the machine
- **`compact`** - If `True`, then if `verbose = True`, print a one-line description of the step; if `False`, print out the recursive substructure of the state update at each step
- **`printInput`** - If `True`, then if `verbose = True`, print the whole input in each step, otherwise don't. Useful to set to `False` when the input is large and you don't want to see it all.

step(self, inp)

Execute one 'step' of the machine, by propagating `inp` through to get a result, then updating `self.state`. Error to call `step` if `done` is true.

Parameters:

- `inp` - next input to the machine

transduce(self, inps, verbose=False, traceTasks=[], compact=True, printInput=True, check=False)

Start the machine fresh, and feed a sequence of values into the machine, collecting the sequence of outputs

For debugging, set the optional parameter check = True to (partially) check the representation invariance of the state machine before running it. See the documentation for the `check` method for more information about what is tested.

See documentation for the `start` method for description of the rest of the parameters.

Parameters:

- `inps` - list of inputs appropriate for this state machine

Returns:

list of outputs

run(self, n=10, verbose=False, traceTasks=[], compact=True, printInput=True, check=False)

For a machine that doesn't consume input (e.g., one made with `feedback`, for `n` steps or until it terminates.

See documentation for the `start` method for description of the rest of the parameters.

Parameters:

- `n` - number of steps to run

Returns:

list of outputs

doTraceTasks(self, inp, state, out, debugParams)

Actually execute the trace tasks. A trace task is a list consisting of three components:

- `name`: is the name of the machine to be traced
- `mode`: is one of `'input'`, `'output'`, or `'state'`
- `fun`: is a function

To **do** a trace task, we call the function `fun` on the specified attribute of the specified mahine. In particular, we execute it right now if its machine name equals the name of this machine.

check(thesm, inps=None)

Run a rudimentary check on a state machine, using the list of inputs provided. Makes sure that getNextValues is defined, and that it takes the proper number of input arguments (three: self, start, inp). Also print out the start state, and check that getNextValues provides a legal return value (list of 2 elements: (state,output)). And tries to check if getNextValues is changing either self.state or some other attribute of the state machine instance (it shouldn't: getNextValues should be a pure function).

Raises exception 'InvalidSM' if a problem is found.

Parameters:

- `thesm` - the state machine instance to check

- `inps` - list of inputs to test the state machine on (default None)

Returns:
none